A Conductible Virtual Orchestra

MEng Project Final Report

Osemwaro Pedro

E-mail: op02@doc.ic.ac.uk

Web page: http://www.doc.ic.ac.uk/~op02/project

Supervisor: Prof. Duncan Gillies

Second Marker: Prof. Guang-Zhong Yang

2nd June 2006

Department of Computing

Imperial College London

"The conductor's baton technique is the means by which he communicates with the players and singers under his direction. It is not an end in itself and must become so automatic that he can concentrate on musical interpretation, rather than on matters of technique."

– J. Lumley and N. Springthorpe, [JLNS1].

Abstract

A global pandemic of rhythmic restlessness has struck regular concert-goers the world over. This illness typically manifests itself in a desire to conduct recordings of the Berlin Philharmonic, or some other renowned orchestra, from one's living room sofa. Many who fall foul to this disease are left with the bitter aftertaste of unrequited love however, as the lack of interactivity in most of today's music playback devices precludes any response from the music.

This report details the development of an interactive, vision-based system that addresses this problem, providing an application through which amateur conductors can improve their conducting technique. The focus is on determining the conductor's [potentially varying] tempo and playing back the music in time with it. Achieving this requires baton tracking, beat detection and beat prediction.

A suitable tracking algorithm for this application must be able to track the baton in real time against a cluttered environment. We will explore the use of the condensation algorithm as a potential baton tracker. To evaluate its accuracy, we will compare its performance to that of a shape recognition-based tracking algorithm that performs well in uncluttered environments. We will also investigate methods for detecting beats after they have occurred and predicting the time at which the next beat will occur.

ACKNOWLEDGEMENTS

I would first of all like to thank my supervisor, Prof. Duncan Gillies, for the time he has invested in this project, his continued support, his insight into the more challenging aspects of the project and his suggestions. I would also like to thank my second marker, Prof. Guang-Zhong Yang, for his time and extra suggestions.

Secondly I'd like to thank the Computing Support Group for providing: the Philips ToucCam Pro II webcam; a dedicated project machine that was conveniently located near a large white wall, which created a suitable test environment; technical assistance with various hardware-related issues.

Thirdly I'd like to thank the staff of the Foundry, and in particular my old manager Andrew Whitmore, for giving me a thorough grounding in the ins and outs of C++ and Visual Studio during my internship with them. The skills I learnt there proved to be exceptionally helpful as far as the implementation of this project was concerned.

I'd also like to thank Christopher Braime for introducing me to [JLNS1] and for pointing out a source of [relatively] cheap batons (that have a habit of breaking when you least expect it!), and also Richard Dickins for providing other useful information on conducting.

Last but not least I'd like to thank my family and friends who've kept me laughing throughout my four-year maiden voyage through the world of higher education.

Contents

PART I: INTRODUCTION

Of all the members of an orchestra, the conductor is the one person who is unable to work on his technique at any time he wishes; not even the most charismatic of conductors have whole orchestras available to them for practise any time of the day or night. This can be an even bigger problem for beginner conductor's who may not yet have an orchestra to conduct at all.

Although basic baton motions can be practised alone (e.g. beating time in front of a mirror), doing so is of limited benefit, as a conductor's development needs to be guided by the invaluable feedback of an orchestra (an inexperienced conductor may be unable to judge how clear his gestures will appear to an orchestra). Having said that, high quality private practise and study is essential for all musicians, including the conductor. Certain tools, such as the metronome and tuning forks, have gained wide acceptance as useful practise aids. The development of a tool to improve the quality of a conductor's private technical practise could prove to be similarly beneficial.

This report focuses on the development of such a tool. More specifically, it deals with the design and creation of a system that we have called a Conductible Virtual Orchestra; a computer program that is able to "watch" how a conductor conducts a piece of music, interpret his gestures as indications of the tempo he desires and then play back the music in synchronisation with this tempo.

A number of other researchers have developed similar projects in the past, many of which required the use of expensive, unwieldy hardware. Some of these are discussed in section II.1. For such a system to have widespread appeal amongst amateur/student conductors however, a low-cost approach would be vital. So for this project we focused on the use of a webcam as the means by which the program can sense the conductor's motion, as webcams are generally very cheap (even for students!).

The idea of sensing the conductor's motion through images captured by a camera falls into a rapidly growing area of Computer Science known as Computer Vision. This area deals with the formidable task of endowing computers with the ability to interpret images, an ability that comes naturally to us humans that we take for granted. Section II.3 of this report presents some of the theoretical material from this area that is relevant to our project. The ways in which we used and further developed that material in order to realise the goals of this project are discussed in section III.2.

Part II: Background

This part of the report introduces the background material needed to implement the system, starting with a summary of some related work. Where an in-depth report of a particular subject area would go out of the scope of the report, references to more detailed sources are given.

II.1 Related Work

A variety of approaches to computer-based conducting systems have been taken by various researchers to date. The following gives a brief summary of some of these systems, and explains how their success and shortcomings have influenced the approach that we we have decided upon for our system.

One of the earliest developed computer conducting systems is M. V. Mathews' Radio Baton and Conductor Program, described in [MVM1]. This system is based on two batons with foam-covered radio transmitters attached to the ends. By measuring the strength of the radio signal at five receiving antennae, the system is able to track the position of the batons in 3D space. Baton movements cause MIDI triggers to be sent to a synthesiser, allowing the user to control the tempo, dynamics and other aspects of the predetermined composition that the synthesiser is playing.

Numerous other systems have since been developed that use sensor-based hardware devices to track the conductor's gestures, including [MLGGDW1] and [JBWSMM1], which use the Buchla Lightning infrared baton ([DB1]). The system described in [JBWSMM1] is an interactive exhibit for a music exhibition, and so was targeted at amateur users who have no knowledge of professional conducting gestures. It is of note though because it transformed recorded audio and video data of a real orchestra playing in order to follow the user's tempo rather than using synthesised sounds (the audio data was time-stretched using the Fourier transform and the video data was synchronised by skipping or repeating frames as necessary).

Other sensor-based approaches are described in [TITT1] and [TMN1], which both use body-sensor-equipped jackets to measure the motion of the conductor's limbs. In addition to this, the jacket used by T. M. Nakra in [TMN1] also measures muscular tension in the arms, heart rate, respiration, skin conductance and body temperature. Nakra's analysis of

the results revealed the importance of respiration and muscular force in communicating musical phrasing and intensity/loudness.

The advantage of using such sensor-based devices in a computer conducting system is that they are able to produce accurate 3D measurements quickly. Efficient computation is critical to such systems as they must operate in real-time, analysing streams of data sampled many times per second. One of the main disadvantages however is that in some cases they can restrict the freedom of the conductor. Concerning the use of a digital baton in an earlier project, Nakra says in [TMN1]:

> "The baton's size and heaviness were not conducive to graceful, comfortable gestures; it was 5-10 times the weight of a normal cork-and-balsa wood conducting baton. A typical 45-minute gestural *Brain Opera* performance with the 10-ounce *Digital Baton* was often exhausting. This also meant that I couldn't take it to orchestral conductors to try it out; it was too heavy for a conductor to use in place of a traditional baton."

Another disadvantage is that the cost of developing such devices can potentially be quite high; the target users of my system, beginner conductors, aren't likely to be willing to spend large amounts of money in order to use it.

For our application we decided to take a vision-based approach similar to D. Murphy's in [DMTAKJ1], in which the motion of the conductor's baton is tracked in the video output of a webcam. The advantage of developing this kind of system is that it wouldn't interfere with the conductor's movements, nor would it be beyond the price range of the average conducting student. The major disadvantage is the difficulty of designing an efficient and robust tracker. Murphy reports attaining a tracking rate of 25 frames/sec in [DM1] at a resolution of 160x120 on a relatively low-spec machine, demonstrating that good results are attainable by this method. Another thing to note is that tracking would produce less accurate estimates of the baton's position than some of the above-mentioned methods, which may make the gesture analysis stage of the application more difficult.

Murphy's system allows for two cameras to be used, one facing the conductor and one to his side. To reduce financial and computational costs however, we just used one camera. This circumvents the technical problem of synchronising and calibrating two cameras. Furthermore, by placing the camera to the side of the conductor rather than in front of him as in Murphy's single-camera system, we reduce the variation in the length of the projection of the baton onto the camera, and we eliminate the problem of the baton "disappearing" when it points straight at the camera. A profile-view camera placement also simplifies tracking, as the side-to-side motion of the baton is less pronounced when viewed from the side. This is a valid position for the camera to be in as it has the same view of the conductor here that the 1st violin section or the 'cello section of a symphony orchestra have.

II.2 The Art of Conducting

This section gives a brief overview of conducting technique. Conducting is a highly expressive art, requiring years of study and practise to master, and so a full treatment of the subject would of course go outside the scope of this report. We will therefore only present the aspects of conducting that are directly related to this project. The reader is referred to [JLNS1], from which the source material for much of this section was taken, for a more detailed guide to conducting.

II.2.1 The Conductor

[JLNS1] identifies the two basic requirements that a conductor must fulfil as "know[ing] the music in great detail and [having] the technique to communicate this knowledge". The authors go on to say in chapter 3 that "communication is achieved by arm and wrist movements and by changes in posture and facial expression, the most important element being the movement of the right arm, which indicates the tempo and dynamics of the music." It is thus the aspects of communication pertaining to the motion of the right arm that this project is concerned with. In particular we will concentrate on the way a conductor communicates the tempo of a piece to the orchestra.

II.2.2 Beating Time

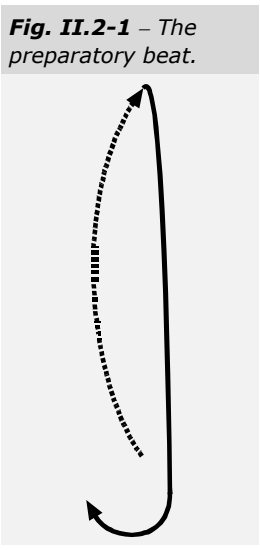


Fig. II.2-1 *– The preparatory beat.*

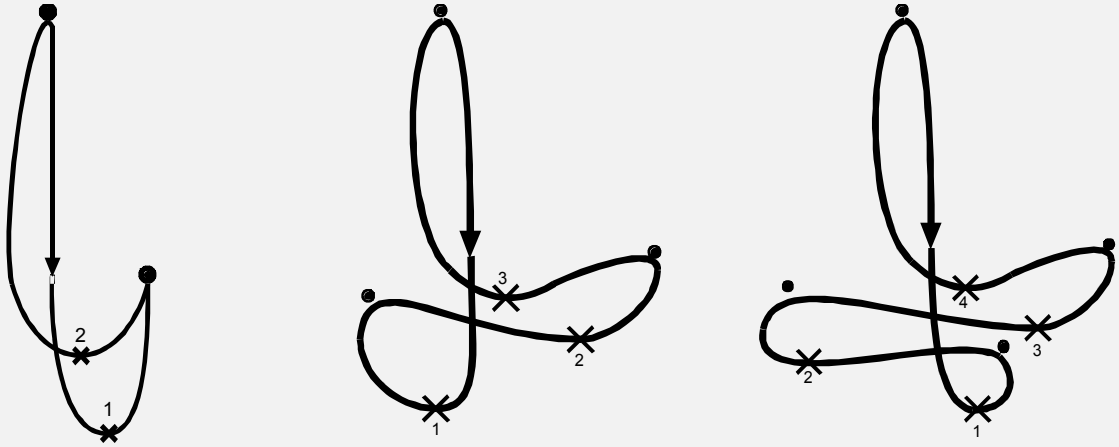
The beat of a piece of music is a regular pulse that underlies the rhythmical variations of each bar. As such it determines the tempo of the music, and so it is critical for each member of the orchestra to be aware of this pulse in order for them to play together. There is an ambiguity inherent in most pieces of music in that multiple pulses can often be felt, each with a frequency that's a multiple of the others. To resolve this, the conductor will usually conduct the pulse that's most comfortable for himself and the orchestra.

It is very important for the orchestra to know what tempo the conductor will start conducting the piece of music with and precisely when he wants the piece of music to start. To communicate this, conductors conduct a preparatory up-down beat one beat before the orchestra should start (see fig. II.2-1). The upwards motion indicates the duration of half a beat. From this the orchestra can predict when the downwards motion will end, which is the point at which they should come in. The horizontal directions of the upwards and downwards motions vary depending on the beat of the bar that the music

starts on.

After the preparatory beat, the tempo is indicated to the orchestra by means of a periodical right-arm gesture that indicates the point in time at which each beat, sub-beat or some multiple of the beat occurs. Although each individual conductor may have personal stylistic preferences as to the precise form of these gestures, there are universally accepted general forms that are common amongst most conductors. Figs. II.2-2 (a)–(c) illustrate the gestures for conducting 2, 3 or 4 beats per bar. The numbered crosses indicate the *ictus*, the point at which each beat occurs, the numbers indicate the beat and the circles indicate the half beats. Note that the ictus always occurs when the baton changes from a downwards motion to an upwards motion. It's also important to note that the major downwards motion always coincides with the first beat of the bar.

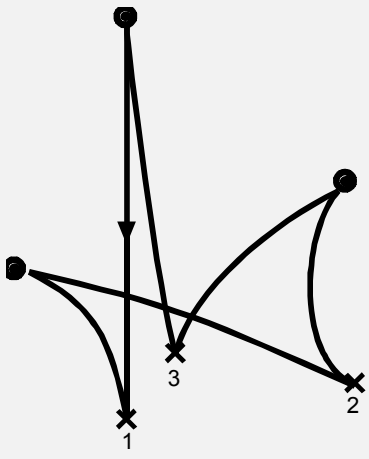
Fig. II.2-2 *– The patterns for conducting 2, 3 or 4 beats per bar, as seen from the conductor's point of view.*



(a) 2 beats (b) 3 beats (c) 4 beats

Variations on these patterns can be used to communicate other things. For example the smoothness of the path traced by the baton describes whether the music should be played *legato* (smoothly) or *staccato* (detached). Fig. II.2-3 illustrates an example of conducting staccato. The *dynamics* (loudness) of the music is indicated by the size of the pattern; small gestures for quiet moments, larger ones for louder moments.

Fig. II.2-3 *– Beating 3 beats per bar staccato.*



II.3 Tracking

A vast array of techniques for tracking the motion of objects have been proposed over the years. The success of any given technique is often highly dependent on the problem at hand; a method that works well in one scenario won't necessarily work as well in another.

The two common general approaches to tracking that are most appropriate to this project are blob tracking and contour tracking. The former involves segmenting the images of a sequence and determining which segments (blobs) correspond to the object of interest. The latter involves tracking the motion and, if appropriate, the deformation of a particular curve through a sequence of images.

For our application we considered two different algorithms for tracking the motion of the conductor's baton. The first algorithm, a blob tracker, uses a statistical shape recognition technique to identify which extracted segment in the image corresponds to the baton. The second algorithm is a contour tracker known as the condensation algorithm (see [MI1]). It works by propagating a conditional probability distribution that represents our beliefs about the state of the baton through time.

Sections III.2.1 and III.2.2 of this report describe our implementation of these algorithms in detail. The next two sections present the elementary concepts of computer vision and statistics that the algorithms are based on.

II.3.1 Image Processing

When tracking an object or performing any other machine vision task, it is often necessary to process the input images in order to reduce noise and other artifacts, and to extract useful information that will be subjected to further analysis later on. This section gives a synopsis of the image processing operations that will be referred to later in this report.

NOTE: throughout the following sub-sections, $f(x)$ and $f(i, j)$ are used to refer to the value of pixels x and (i, j) respectively of an intensity image f. The techniques can generally be applied to multi-channel images by applying them to each channel separately.

II.3.1.1 Convolution

[MWWR1] defines convolution as "an integral that expresses the amount of overlap of one function g as it is shifted over another function f. It therefore 'blends' one function with another". It is the basis of a number of image filtering methods, and is formally defined as follows:

$$(f * g)(x) = \int f(\tau) \cdot (x - \tau) d\tau$$

Or in its discrete form it is:

$$(f * g)(x) = \sum_n f(n) \cdot g(x - n)$$

Where:

$f * g$ is the convolution of function f with g. Function g is sometimes referred to as the kernel function.

II.3.1.2 Noise Filtering

Many different types of filters exist for reducing the amount of noise in an image. The one this report will refer to most is the **Gaussian filter**. It has the effect of blurring an image without introducing visual artifacts such as horizontal/vertical lines that certain other blurring methods create.

In its true form, the Gaussian filter is defined as the convolution of a Bivariate Gaussian distribution with f, where the mean of the distribution is taken as the location of the desired output pixel. A commonly used approximation however is to define a $(2R+1)\times(2R+1)$ matrix G called a convolution filter that has the form of a truncated, discretised Gaussian distribution. The discrete form of the convolution equation can then be used:

$$(f * G)(x,y) = \sum_{a=-R}^{R}\left(\sum_{b=-R}^{R} f(x+a, y+b)\cdot G_{R+b+1,R+a+1}\right)$$

A typical example of G might then be:

$$G = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \text{ for } R=1$$

Another useful blurring operation is the median blur. This operation sets a pixel's intensity to the median intensity of itself and its local neighbours. It has the property of preserving sharp edges rather than blurring them.

II.3.1.3 Edge Detection

Knowing where the edges of objects in an image are is often useful for segmentation and various other operations. Edges are defined as areas of an image where there is a sudden change in intensity. We can use convolution filters to find such points, e.g. the Sobel operator:

$$Gx` = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad Gy` = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$Gx(p) = (f * Gx`)(p) \qquad Gy(p) = (f * Gy`)(p)$$

$$|G(p)| = \sqrt{Gx(p)^2 + Gy(p)^2}$$
$$\theta(p) = \arctan\left(Gy(p) \big/ Gx(p)\right)$$

Where:

$|G(p)|$ is a measure of the strength of the edge at pixel p.

$\theta(p)$ gives the orientation of the edge at pixel p.

II.3.1.4 Segmentation

Segmentation is the process of partitioning an image into distinct regions that correspond to individual objects. Pixels in an image might be grouped into the same segment if they:

- are of similar luminance (i.e. if they aren't separated by an edge) ,
- are of similar colour,
- define an area of uniform texture,
- match the known hue and saturation histograms of the object(s) we're interested in,
- match the known texture of the object(s) we're interested in.

One way of finding all the pixels in the same segment as a given pixel is to use a flood fill algorithm to iterate over its neighbours, using some of the above criteria (or some other criteria) to determine where the boundaries lie.

II.3.1.5 Background Subtraction

The goal of background subtraction is to identify and mask out pixels in an image that are part of the background, so that pixels corresponding to the foreground objects can be identified.

The simplest background subtraction method for image sequences with a static background would be to define an image B of the background on its own, and then classify all pixels of f that are similar to the corresponding pixel in B as background pixels. Two pixels can be deemed to be similar if their absolute difference is less than some threshold.

This method would almost certainly fail however as it wouldn't be robust to noise. An improvement on this method is to calculate B as a running average of background images over an arbitrary number of frames, so that B can be taken as the expected background image. Pixels of f that are classified as background pixels can be used to update the corresponding pixels of B, however the potential for misclassification could lead to a degradation in the accuracy of B.

The reader is referred to [AMM1] for a review of more background subtraction techniques that have been published in the literature, such as the adaptive mixture of Gaussians method.

II.3.2 Shape Recognition

The techniques outlined in the previous section allow us to extract primitive features from an image such as the pixels that lie on an edge or segments that correspond to objects (or part of an object). We must process these features at a higher level in order to determine more useful information about the image, such as whether or not the image contains an object of interest.

Suppose we have extracted a set P of 2D coordinates that correspond to the pixels of an image segment. To determine whether or not P represents the shape we are interested in, we need to evaluate it with respect to a model of our prior knowledge of that shape.

One common approach to this is to define a metric $M(P)$ that represents how well P corresponds to our model. Any segment P' for which $M(P')$ has a low value can be discarded as an unlikely candidate. Depending on our requirements, we may then take all segments for which $M(P)$ > some threshold T as the desired objects, or we may take the segment or segments that maximise $M(P)$.

Fig. II.3-1 *– A typical image of a black baton on a white background.*

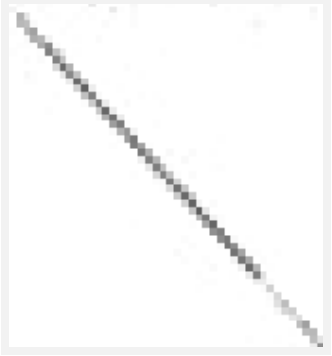


Fig. II.3-1 shows the typical appearance of the baton. A metric that determines how similar a given segment is to a baton should take into account the segment's length, thinness and straightness. The length of a segment P is given by the Euclidean distance between the two pixels p_0 and p_1 of P that project onto the outermost extremities of P's central axis. The standard deviation of pixels about this axis can be taken as a measure of P's thinness. By only considering segments that are sufficiently thin as candidate batons, we implicitly force the candidates to be straight.

Given the length L_P of segment P and standard deviation σ_P of its pixels about its central axis, we can define $M(P)$ as:

$$M(P)=\begin{cases} L_P=\|p_1-p_0\|, & \sigma_{min}\le\sigma_P\le\sigma_{max}, L_{min}\le L_P\le L_{max} \\ 0, & otherwise \end{cases}$$

Where:

σ_{min}, σ_{nax}, L_{min}, L_{max} define the min/max permissible values of σ_P and L_P.

Given p_0 and p_1, we can use the fact that the baton's tip will usually be closer to the edge of the frame that the conductor is facing to determine which ends of the baton p_0 and p_1 correspond to (the user can specify what direction the conductor is facing). If P's central axis is near-vertical, the point furthest from the horizontal line that runs through the middle of the frame can be taken as the tip.

To measure L_P and σ_P from P, we first of all need to determine P's central axis. We can use regression to do this.

II.3.2.1 Linear Regression

Given a set of data $P=\{(x_0, y_0), \ldots, (x_{N-1}, y_{N-1})\}$ (the (x, y) terms can in general be scalars or vectors), regression in its most general form can be defined as the process of finding an optimal parameterisation of a function f such that:

$$y_i = f(x_i, \Xi) + \epsilon_i, \quad 0 \le i < N$$

Where:

$\epsilon_0, \ldots, \epsilon_{N-1}$ are error terms to be minimised by the optimisation process.

Ξ is the set of f's parameters.

A commonly used optimisation scheme is the least squares method, in which we find Ξ as follows:

$$\Xi = \arg\min_{\Xi'} \sum_{i=0}^{N-1} \epsilon_i^2 \equiv \arg\min_{\Xi'} \sum_{i=0}^{N-1} (y_i - f(x_i, \Xi'))^2 \qquad \text{(Eq. II.3.2.1)}$$

To find the central axis of P, we require a linear function $f(x) = a + bx$, where $\Xi = (a, b)$. Each ϵ_i is then equal to $y_i - f(x_i)$ = the vertical distance from (x_i, y_i) to $(x_i, f(x_i))$. The closed-form solution to eq. II.3.2.1 when f is of this form can be found by calculating the values of a and b for which $\frac{d}{da}\left(\sum_{i=0}^{N-1} \epsilon_i^2\right) = 0$ and $\frac{d}{db}\left(\sum_{i=0}^{N-1} \epsilon_i^2\right) = 0$. The solution is given in [MWWR2]. However f is undefined when the central axis is a vertical line. One solution to this problem is to use polar coordinates.

II.3.2.2 Polar Coordinate Linear Regression

To solve the regression problem using polar coordinates, we must express the problem in vector form. We seek a vector L:

$$L(\mu; \bar{p}, \theta) = \bar{p} + \mu(\cos\theta, \sin\theta)$$

s.t.
$$(x_i, y_i) = L(\mu_i; \bar{p}, \theta) + \epsilon_i$$

and
$$(\bar{p}, \theta) = \arg\min_{(\bar{p}', \theta')} \left[\sum_{i=0}^{N-1} \|\epsilon_i\|^2\right] = \arg\min_{(\bar{p}', \theta')} \left[\sum_{i=0}^{N-1} \|(x_i, y_i) - L(\mu_i; \bar{p}', \theta')\|^2\right] \qquad \text{(Eq. II.3.2.2)}$$

Where: $\mu_i = ((x_i, y_i) - \bar{p}) \cdot (\cos\theta, \sin\theta)$ = projection of $(x_i, y_i) - \bar{p}$ onto $(\cos\theta, \sin\theta)$.

Note: θ is measured anticlockwise from the x-axis (see Fig. A-1).

Each ϵ_i error term now defines the perpendicular vector from (x_i, y_i) to L. The parameters that we must find are $\bar{p}$ and θ. Theorem II.3.2.2-1 gives a closed-form solution to this problem.

Calculating the parameters using theorem II.3.2.2-1 involves calculating a function $R(\cos\theta, \sin\theta)$, which gives the sum of the squared perpendicular distances between vector L and the elements of P. Hence, the standard deviation σ_P of the elements of P about central axis L is given by $\sigma_P = \sqrt{\frac{R(\cos\theta, \sin\theta)}{N-1}}$.

To find p_0 and p_1, the outermost extremities of segment P, we transform P's elements in such a way that their central axis lies on the x-axis, take the leftmost and rightmost transformed elements, and then apply the inverse transformation to transform them back to the original coordinate space. The transformation is simply a translation by $^{-}\bar{p}$ followed by a clockwise rotation by θ radians.

Theorem II.3.2.2-1 *– Closed-form solution to the polar coordinate linear regression problem.*

Given a set $P = \{(x_0, y_0), \ldots, (x_{N-1}, y_{N-1})\}$, the line $L(\mu; \bar{p}, \theta) = \bar{p} + \mu(\cos\theta, \sin\theta)$ satisfies eq. II.3.2.2, i.e. minimises the sum of the squared perpendicular distances between itself and the elements of *P*, where the parameters are calculated as follows:

$$\bar{p} = (\bar{p}_x, \bar{p}_y) = \frac{1}{N}\left(\sum_{i=0}^{N-1} p_{ix}, \sum_{i=0}^{N-1} p_{iy}\right)$$

$$(\cos\theta, \sin\theta) = \arg\min_{\omega \in \Omega} R(\omega)$$

Where:

$$p_i = (x_i, y_i)$$

$$v_i = p_i - \bar{p}$$

$$\Omega = \{(c, s), (-c, s), (s, c), (-s, c)\} \subseteq \mathbb{R} \times \mathbb{R}$$

$$R : \Omega \to \mathbb{R}$$

$$R(\omega) = \sum_{i=0}^{N-1} \left[v_i \cdot v_i - (v_i \cdot \omega)^2\right]$$

$$c = |\cos(\phi)| = \left|\sqrt{\frac{|\cos(2\phi)| + 1}{2}}\right|$$

$$s = |\sin(\phi)| = \left|\sqrt{1 - c^2}\right|$$

$$|\cos(2\phi)| = \left|\frac{A}{\sqrt{A^2 + B^2}}\right|$$

$$A = \frac{1}{2}\sum_{i=0}^{N-1}\left(v_{iy}^2 - v_{ix}^2\right)$$

$$B = \sum_{i=0}^{N-1} v_{ix} \cdot v_{iy}$$

The proof of this is given in Appendix A. Note that by expanding the expressions for *A*, *B*, and *R*(…), it is possible to calculate $\bar{p}$ and $(\cos\theta, \sin\theta)$ in a single pass over the data. See section III.2.1.4 for the details of this expansion.

II.3.3 The Condensation Algorithm

The condensation algorithm belongs to a class of algorithms known as particle filters. Particle filters provide a simulation-based approach to estimating the parameters of some model from observed data. In the case of this project, we are interested in a model of the motion of the baton. The parameters that we need to estimate define the baton's state (i.e. its position, velocity and acceleration) at any given time.

The formulation of the condensation algorithm that we used as a reference ([MI1]) was designed for the near-real-time tracking of curves in a cluttered environment. Its main advantage over other tracking algorithms (such those based on the Kalman filter) are that it is able to track objects through multi-modal environments, where there are multiple potential candidates for the single object we're trying to track.

The following sections explain the theoretical basis of the algorithm. As in [MI1], we will begin by considering the problem of finding an object in a static image, after which we will show how the technique can be extended to tracking an object in a video sequence.

II.3.3.1 Factored Sampling

Suppose we have a set Z of data extracted from an image (e.g. by any of the means described in section II.3.1) and we want to use this data to determine the state X of an object in that image. Let $P(X)$ be the prior probability of the object being in a particular state. By Bayes' rule we have:

$$P(X|Z) = k \cdot P(Z|X) \cdot P(X) \qquad \text{(Eq. II.3.3.1)}$$

Where:

k is a normalising constant.

The technique of factored sampling can be used to estimate $P(X|Z)$, the posterior probability of the object being in state X given Z. First of all we generate a sample set $S=\{s_0, s_1, \ldots, s_{N-1}\}$ for some N from the prior distribution, $P(X)$. We then define a random variable X' with the following probability distribution conditional on Z:

$$P(X' = s_i \mid Z) = \frac{P(Z|X = s_i)}{\sum_{j=0}^{N-1} P(Z \mid X' = s_j)} \qquad \text{(Eq. II.3.3.2)}$$

$P(X'=s_i \mid Z)$ approximates $P(X|Z)$ with increasing accuracy as N increases. The expected value of some function $f(X)$ can then be estimated as:

$$E[f(X)\,|\,Z] \approx E[f(X')\,|\,Z] = \sum_{i=0}^{N-1} f(s_i)\cdot P(X' = s_i\,|\,Z) \qquad \text{(Eq. II.3.3.3)}$$

Thus for $f(X)=X$, $E[f(X')\,|\,Z]$ approximates the mean state of the object, which we take as its state in the image.

II.3.3.2 Conditional Density Propagation

The condensation algorithm extends the factored sampling technique in order to track the motion of an object through a sequence of images by propagating the distribution of the posterior $P(X'\,|\,Z)$ through time. We denote the posterior for time t as $P(X_t\,|\,\mathcal{Z}_t)$, where X_t is the state of the object at time t, $\mathcal{Z}_t = \{Z_0, Z_1, \ldots, Z_t\}$ and each Z_i denotes the data extracted from the image that occurred in the sequence at time i.

Our estimation of the prior is based on the following equation:

$$P(X_t\,|\,\mathcal{Z}_t) = k_t\cdot P(Z_t\,|\,X_t)\cdot P(X_t\,|\,\mathcal{Z}_{t-1}) \qquad \text{(Eq. II.3.3.4)}$$

Where:

k_t is a normalising constant.

The prior, $P(X_t\,|\,\mathcal{Z}_{t-1})$, is defined as:

$$P(X_t\,|\,\mathcal{Z}_{t-1}) = \int P(X_t\,|\,X_{t-1})\cdot P(X_{t-1}\,|\,\mathcal{Z}_{t-1})\, dX_{t-1} \qquad \text{(Eq. II.3.3.5)}$$

We sample from the prior $P(X_t\,|\,\mathcal{Z}_{t-1})$ by generating a sample s' from $P(X_{t-1}\,|\,\mathcal{Z}_{t-1})$, the posterior distribution from time *t-1*, and then making a prediction conditional on s' from the motion model $P(X_t\,|\,X_{t-1}=s')$. At time *t=1* however, $\mathcal{Z}_{t-1} = Z_0 = \emptyset$, and hence $P(X_t\,|\,\mathcal{Z}_{t-1}) = P(X_t)$, so we simply sample directly from $P(X_t)$ in this case, which handles the initialisation of the tracker.

The algorithm is given in alg. II.3.3.2-1 below. The reader is referred to [MI1] for proof of the correctness of the above equations and the algorithm. It is based on the following assumptions:

- Let $\mathcal{X}_t = \{X_0, X_1, \ldots, X_t\}$.
- The object's state at time t is only dependent on its state at time *t-1*. This motion model is expressed by the following first-order Markov chain:

$$P(X_t\,|\,\mathcal{X}_{t-1}) = P(X_t\,|\,X_{t-1})$$

- The observed image data at each time step t is independent of: the observed image data at all other time steps; the motion model, i.e.:

$$P(\mathcal{Z}_{t-1}, X_t \mid \mathcal{X}_{t-1}) = P(X_t \mid \mathcal{X}_{t-1}) \cdot \prod_{i=1}^{t-1} P(Z_i \mid X_i)$$

Alg. II.3.3.2-1 *– The Condensation Algorithm: this iteration of the algorithm propagates our estimate S_t of the posterior $P(X'_{t-1}|\mathcal{Z}_{t-1})$ from time step t-1 to time t.*

– From time step t-1 we have sample set $S_{t-1} = \{<s_i, P(X_{t-1}=s_i|\mathcal{Z}_{t-1})>; 0 \leq i < N\}$

– We need to calculate S_t:

$S_t \longleftarrow \phi$

For each i, $0 \leq i < N$:

Select an element $<s_i, p_i>$ from S_{t-1} with probability p_i.

Predict a new sample s_i' by sampling from $P(X_t \mid X_{t-1}=s_i)$.

$p_i' \longleftarrow P(Z_t \mid X_t = s_i')$.

$S_t \longleftarrow S_t \cup \{<s_i', p_i'>\}$.

Normalise all of the p_i' probabilities in S_t.

Output $E[X_t \mid Z_t]$ as the mean state of the object at time t.

II.3.3.3 Importance Sampling and (Re)initialisation

Importance sampling provides us with a way to reduce the variance of the samples in the set S_t generated by the conditional density propagation method. This has the effect of reducing the size of the sample set needed to track an object with a given degree of accuracy, thus improving the tracker's efficiency.

The idea is to introduce a distribution $g(X_t)$ (sometimes called an "importance function") that is biased towards the most likely values of X_t. Rather than drawing samples from the posterior from the previous iteration and making a prediction from these using our motion model, we draw samples from $g(X_t)$ and then weight the likelihood so as to make our estimate of the posterior unbiased. Thus after drawing a sample s' from $g(X_t)$, we add the element $<s', w' \cdot P(Z_t \mid X_t=s')>$ to S_t, where w' is calculated as follows:

$$w' = \frac{\sum_{j=0}^{N-1} P(X_{t-1}=s_j \mid \mathcal{Z}_{t-1}) \cdot P(X_t=s' \mid X_{t-1}=s_j)}{g(X_t=s')} \qquad \text{(Eq. II.3.3.6)}$$

Where:

$<s_j, P(X_{t-1} \mid Z_{t-1})> \in S_{t-1}$, $0 \leq j < N$.

Notice that the numerator in this equation is a discretisation of eq. II.3.3.5. Thus

$w' \cdot P(Z_t \mid X_t = s') \approx P(X_t = s' \mid \mathcal{Z}_{t-1}) \cdot P(Z_t \mid X_t = s') \cdot (g(X_t = s'))^{-1}$. This gives us an unbiased estimate of the posterior $P(X_t{=}s' \mid \mathcal{Z}_t)$ as the probability of drawing sample s' is $g(X_t = s')$ (see [WP1] for more details).

To (re)initialise the system we simply take $g(X_t)$ as the prior distribution and sample from it. Since (re)initialisation ignores the previous state of the system, the original factored sampling posterior estimator (eq. II.3.3.2) should be used to calculate the posterior in this case.

II.4 Sampling Probability Distributions

Stochastic tracking algorithms, such as the condensation algorithm, rely heavily on the idea of randomly sampling probability distributions. The following sections explain how we can sample from the three main types of probability distribution that are relevant to this project: uniform distributions, empirical distributions and multivariate Gaussian distributions.

II.4.1 Sampling Uniform Distributions

The uniform distribution forms the basis of most other probability distribution sampling algorithms, including those mentioned above. Hence if our uniform distribution sampling algorithm doesn't generate samples that are in some sense representative of a uniform distribution, we will not be able to use those samples to generate samples that are representative of any other distribution. Thus before adopting a particular PRNG (Pseudo Random Number Generator), it is important to consider whether or not the statistical properties of its output will be satisfactory for the task at hand.

One method for testing how uniformly distributed the output of a PRNG is is the *k*-distribution test. [AKCDL1] defines a sequence of random variables X_0, X_1, X_2, ... $\in$ [0, 1) as being *k*-distributed if:

$$\forall\, n,\, A \in [0,1)^k,\, B \in [0,1)^k,\, \left(P(A_0 \leqslant X_n < B_0, \ldots, A_{k-1} \leqslant X_{n+k-1} < B_{k-1}) = \prod_{i=0}^{k-1} (B_i - A_i) \right)$$

Where:

A_i, B_i are the i^{th} components of the *k*-dimensional vectors *A* and *B*.

In practise no PRNG can be *k*-distributed for *all A*, *B* as required by the definition due to computational limitations. To account for this, the hypercube domain of *A* and *B* is usually quantised as follows:

$$A, B \in Q = \{ i/2^v;\ 0 \leqslant i < 2^v \}^k, \quad v \in \mathbb{N}$$

A PRNG that is *k*-distributed for all *A* and *B* in *Q* is said to be *k*-distributed to *v*-bit accuracy. Clearly the greater the degree of this accuracy, the more uniformly distributed a PRNG's output is.

Another important test to consider is the lag-*k* autocorrelation. This measures how independent the samples generated by a PRNG are. It is defined as:

$$\frac{\sum_i C_{i,k}}{\sum_i C_{i,0}}$$

Where:

$$C_{i,j} = E\left[(X_i - \overline{X}) \cdot (X_{i+j} - \overline{X})\right]$$

It can be shown that the lag-*k* autocorrelation of a uniform distribution is 0 for all *k*. The independence test that arises from this measurement involves determining how close to 0 the lag-*k* autocorrelation is for a particular PRNG over a range of values of *k*.

One PRNG that scores well in both of these tests is the Mersenne Twister generator. [MMTN1] reports that it is 623-distributed to 32-bit accuracy, which is considerably higher than most other PRNGs. Tests performed by [AKCDL1] show that it passes the lag-*k* autocorrelation test for most *k* between 1 and 10. Furthermore it has a colossal period of 2^{19937}-1 (hence the name "Mersenne") and it is highly efficient, as it only uses bitwise operations and additions. All of these factors make it a suitable PRNG for this project. A more detailed discussion of the algorithm along with source code can be found in [MMTN1].

II.4.2 Sampling Empirical Distributions

An empirical distribution for a random variable *X* is a discrete distribution of finite range *R*, for which the probability density function *f*(*X*) has been estimated by experiment. We can sample from such distributions by using the inverse transform method.

The inverse transform method is based on the fact that for any cumulative distribution function *F*(*X*), $P(F(X) \leq u) = u$ for all $0 \leq u \leq 1 \implies F(X) \sim U(0, 1) \implies X \sim F^{-1}(U(0, 1))$. But $X \sim f(X) \implies F^{-1}(U(0, 1)) = f(X)$. Therefore, we can generate samples from any distribution whose cumulative distribution function is known and can be inverted by first of all generating samples from $U(0, 1)$ and then transforming them by the inverse of the CDF. The proof of this result is given in [WP2].

So given an empirical distribution *f*(*X*) where *X* is discrete and ranges over $x_0, x_1, \ldots x_{R-1}$, we can define the CDF *F*(*X*) as $Y_i = F(X \leqslant x_i) = \sum_{j=0}^{j=i} f(x_j)$, for all *i*, $0 \leq i < R$. These Y_i's can be calculated and stored in advance. The inverse function $F^{-1}(Y)$ is then given by a "reverse lookup" approach, i.e.:

$$F^{-1}(Y) = x_j, \ (j = 0 \wedge Y \leqslant Y_j) \vee (0 < j < R \wedge Y_{j-1} < Y \leqslant Y_j)$$

This inverse function can be calculated efficiently (in $O(\log_2 R)$ time) using binary search.

II.4.3 Sampling Gaussian Distributions

The CDF of a Gaussian distribution cannot be expressed in closed form, and so we cannot sample from a Gaussian distribution using the inversion method. There is however another approach known as the Box-Muller transform, which generates a pair of independent, standard normally-distributed random variables X_0 and X_1 from a pair of independent, uniformly-distributed r.v.s U_0 and U_1. [JG1] gives proof of the method, and so we will only state the results here.

The method can be expressed as a pair of closed-form trigonometric equations and as a rejection-sampling based algorithm that uses polar-coordinates (see alg. II.4.3-1). The latter formulation is generally said to be more efficient than the former despite the uncertainty in its termination condition.

Alg. II.4.3-1 *– A rejection sampling-based approach to the Box-Muller transform.*

– Given a pair of independent r.v.s U_0, $U_1 \sim U(-1,1)$, *this algorithm generates a sample from a*
– pair of independent r.v.s X_0, $X_1 \sim N(0, 1)$.

$r \longleftarrow 0$

do:
  $(u_0, u_1) \longleftarrow$ sample from (U_0, U_1)
  $r \longleftarrow u_0^2 + u_1^2$
while: $r=0 \vee r > 1$

Output $(X_0 , X_1) = (u_0, u_1)\cdot(^{-2\cdot(\ln r)}/_r)^{1/2}$

The termination condition depends on the generation of a random point $(u_0, u_1) \in [-1,1]^2$ that falls within the unit circle. Consequently, the number of times the termination condition fails before the algorithm terminates follows a geometric distribution with success parameter 0.25π. From this we can deduce that the average number of iterations is $1+(1-0.25\pi)\cdot(0.25\pi)^{-1} = 1.27$.

Given a sample x generated by this algorithm, we can transform it into a sample x' from the distribution $N(\mu, \sigma)$ using the equation $x' = x\cdot\sigma+\mu$.

II.4.3.1 Multivariate Gaussian Distributions

The multivariate Gaussian distribution is a generalisation of the Gaussian distribution to more than one dimension. A random n-vector $X = (X_0, X_1, ..., X_{n-1})^T$ that follows an n-dimensional multivariate Gaussian is parameterised by its mean n-vector $\mu=(\mu_0 , \mu_1 , \ldots , \mu_{n-1})^T$ and its $n\times n$ covariance matrix $\Sigma = E[(X-\mu)\cdot(X-\mu)^T]$.

In the special case where the off-diagonal covariance values in Σ are all zero, the ele-

ments of X are all mutually independent, and so we can generate a sample from $f(X; \mu, \Sigma)$ by generating n independent samples from $X_i \sim N(\mu_i, \Sigma_{i\,i})$ for each X_i using the Box-Muller transform. In the more general case where the off-diagonal covariance values are non-zero, we first of all use the following fact to sample the multivariate Gaussian n-vector $X' = X-\mu$ with covariance matrix Σ and zero mean:

> *If column vector Y follows an n-dimensional multivariate Gaussian distribution with zero mean and covariance matrix I (the n× n identity matrix), and X'=AY for some n×n matrix A...*

$$\Sigma = E[X \cdot X^{\mathsf{T}}] = E[A \cdot Y \cdot (A \cdot Y)^{\mathsf{T}}] = A \cdot E[Y \cdot Y^{\mathsf{T}}] \cdot A^{\mathsf{T}} = A \cdot I \cdot A^{\mathsf{T}} = A \cdot A^{\mathsf{T}}$$

So if we can find a matrix A such that $\Sigma = A \cdot A^{\mathsf{T}}$, then we can generate a sample x' from X' by first generating a sample y' from Y (whose off-diagonal covariance values are all zero) and then setting $x' = Ay'$. Finally, a sample x from X is given by $x = x'+\mu$.

Matrix A can be generated by the Cholesky decomposition. A full description of this method can be found in [WP3], and an $O(n^3)$ algorithm to calculate it can be found in [NRC1].

PART III: IMPLEMENTATION

The system we have implemented presents a vision-based solution to the problem of responding to a conductor's gestures in real time as he conducts a piece of music. The system is able to interpret the motion of the conductor's baton as an indication of the tempo of the piece of music he is conducting, in response to which it can play back the piece of music (in MIDI form) in synchronisation with him.

The sub-sections of this part contain a detailed discussion of the approach we took in implementing the various components of the system. We will consider how we used the background material presented in the previous part, as well as the ways in which we expanded upon that material in order to achieve the desired results. Before this however, we present an overview of the system's design and the function served by its main components.

III.1 System Design

When implementing any large system, it is easy to fall into the trap of over-complicating its design to the point where its structure becomes incomprehensibly convoluted and impossible to maintain. This tends to be one of the major causes of project failure, and so one of the most critical steps in the development of our system was the creation of a clear, concise design that would allow us to concentrate on achieving the main goals of our project without having to spend too much time trying to correct or work around any limitations caused by structural deficiencies.

III.1.1 Problem Decomposition

The first step in designing the system was to decompose the problem we wanted to solve into a small set of well-defined sub-problems. We identified the following as being central to our solution:

- **Video streaming:** the main input to the system is a video stream of the conductor conducting. The system should be able to accept video streams both from video files (for testing purposes) and from cameras connected to the computer (for interactive performance).

- **Baton tracking:** the system must be able to track and output the position of the baton's tip and base as observed in every frame of the video stream. Depending on the tracking algorithm used, the system may also need to be able to determine additional derivatives of motion such as the velocity and acceleration vectors of the tip and base of the baton.

  The tracking component must be:

  - Robust – it needs to be able to continuously track the motion of the baton for the duration of the piece of music. Realistically speaking, as far as the most likely usage of the system is concerned, this may mean tracking for 3-8 minutes.
  - Insensitive to noise – background noise can easily distract a tracker, which in turn can mislead other components that are directly or indirectly dependent on the tracker's output. The tracker should be able to cope with "reasonable" amounts of noise, detect when it has lost track of the baton and reinitialise itself accordingly.
  - Efficient – section III.3.1 discusses the minimum frame rate needed for the system to be able to follow the conductor at a given tempo. However as a general rule, a higher tracking frame rate allows the beat detection component (see next bullet point) to analyse the baton's trajectory more accurately and to give the times at which it detects beats to a greater degree of precision. Furthermore an efficient tracking algorithm leaves more CPU time for the other components, allowing them to perform more intense computation if necessary.

- **Beat detection:** the trajectory of the baton's tip and base must be analysed in order for the system to be able to determine the time at which each half beat occurs. Depending on the quality of the tracker's output, this component may need to filter the tracked baton trajectory further, so as to avoid false positives (i.e. the mistaken detection of a non-existent half beat).

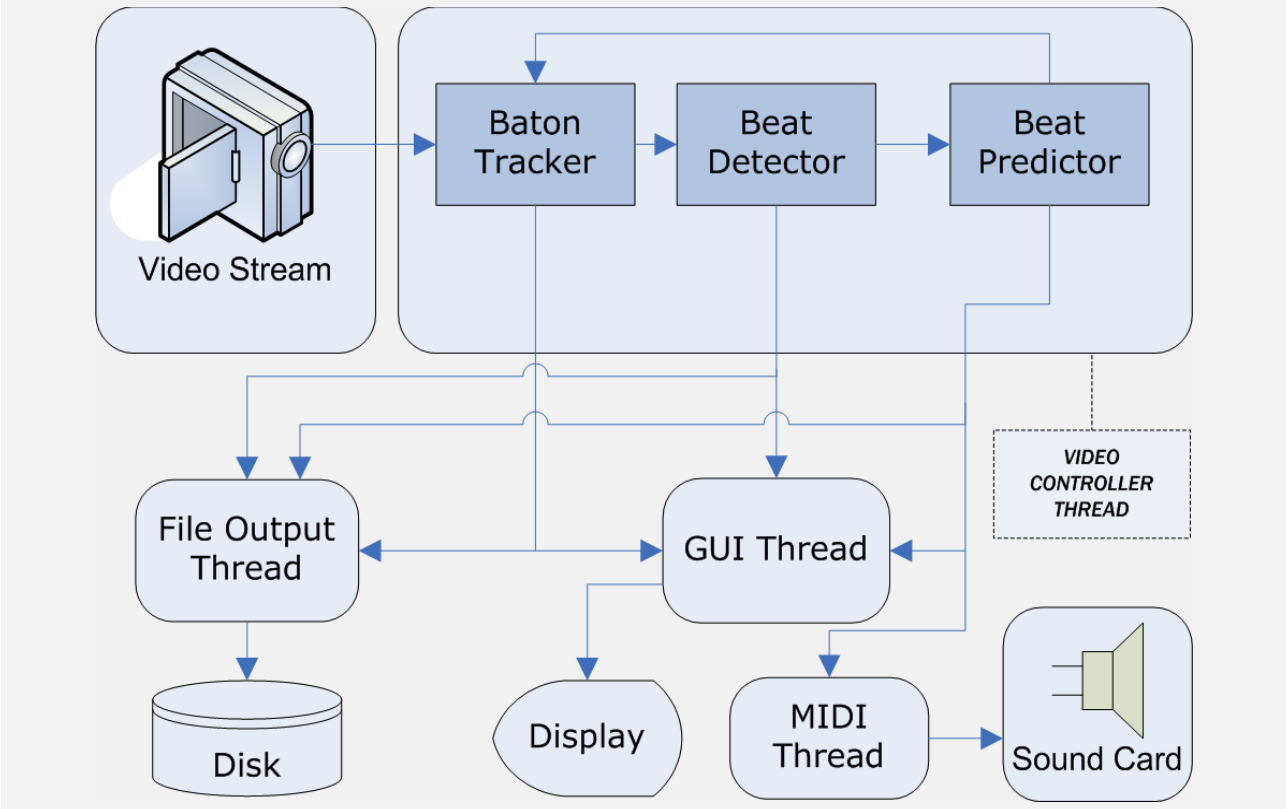
- **Beat prediction:** the beat detection component can only ever recognise the time at which beats occur after they have occurred. By this time it is too late to play the corresponding section of music. To alleviate this problem, the system must be able to make predictions about when future beats will occur.

- **Music playback:** the system must be able to play back the piece of music that the conductor is conducting, varying the tempo in accordance with the half beat times predicted by the beat prediction component.

- **File output:** the various system components should be able to write their output to disk for further analysis at a later time.

There is a clear flow of information from one component to the next in this list, which highlights some of the types of inter-component interaction that our design must take into account. In the visualisation of the information flow given in fig. III.1-1, components are grouped together into a single thread when there is a strictly sequential flow of information between them. Other components, such as the baton tracker and the video stream, need to execute concurrently however, so they are put into separate threads.

Fig. III.1-1 *– Information flow diagram. The curved rectangles represent threads running concurrently with one another, whilst the darker boxes within them represent some of the individual components. The arrows show the flow of data in-between threads and components. Points where an arrow forks represent information flowing to multiple destinations.*



Extra components have been added to show the flow of information to the GUI-updating thread and the file-writing thread (which is used to record system data for later analysis). Notice also that the flow of information between the baton tracker, beat detector and beat predictor forms a loop. The arrow running from the beat predictor to the baton tracker has been added to model the fact that the tracker's behaviour may depend on its belief of when the next beat will occur.

III.1.2 System Components

Having now discussed the overall system design, we can consider the structure of the components in more detail. The key word that summarises our approach to this aspect of the design is "abstraction". Carefully defining interfaces and abstract classes that encapsu-

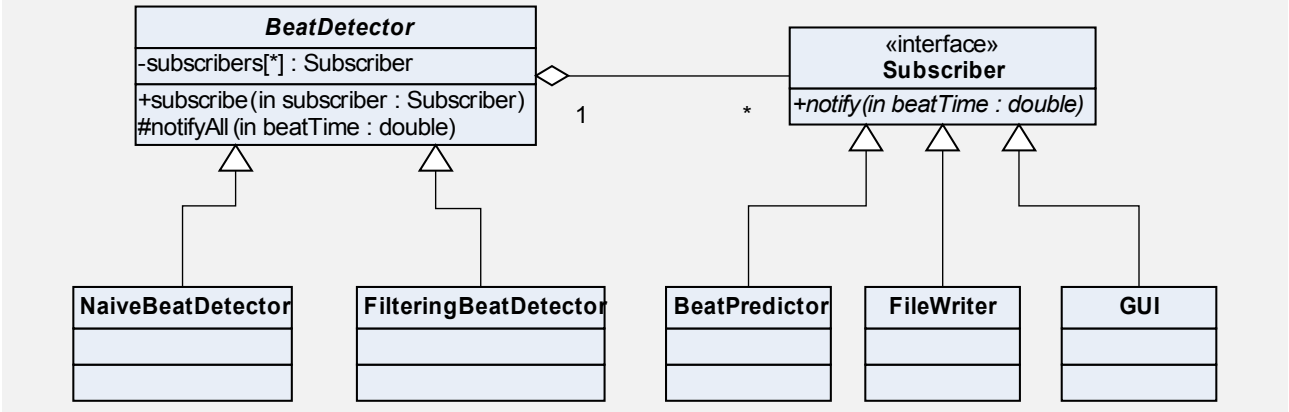
late the desired behaviour of the individual parts whilst hiding implementation-specific details gave us leeway to experiment with multiple implementations of the various components. The following sub-sections highlight a few of the more interesting aspects of the component design.

III.1.2.1 The Beat Detector Interface

The beat detector component periodically sends information to three other components. It would be a waste of CPU time for these three components to poll the beat detector to check whether or not a beat has been detected. Worse still, the GUI and data-logging components may miss some of the detected beats by this method as they run in separate threads to the beat detector.

To avoid these problems we used the publish/subscribe design pattern. In this pattern, the beat detector is viewed as a publisher (of beat times) and the three components that need to receive notifications of each beat time are subscribers. These three components subscribe to the beat detector during the initialisation phase. After this, the beat detector notifies the subscribers through a "notifyAll" method that iterates over all of the current subscribers, calling a notification method declared in the subscriber interface.

Fig. III.1-2 *– UML class diagram showing the publish/subscribe structure of the detected-beat notification procedure.*



III.1.2.2 The Video Processing Interfaces

The video streaming component posed an interesting design challenge, as the two types of video source that this project deals with differ in their frame-access abilities. Video files provide random access to all frames of the video, whereas video cameras can only provide frames in sequential order. Yet it was desirable for us to be able to encapsulate the behaviour of the two types of video source behind a common interface, so that other components in the system would only need to deal with one interface rather than two. The solution we chose was to declare a method in the video stream interface that returns metadata describing the stream's frame searching capabilities.

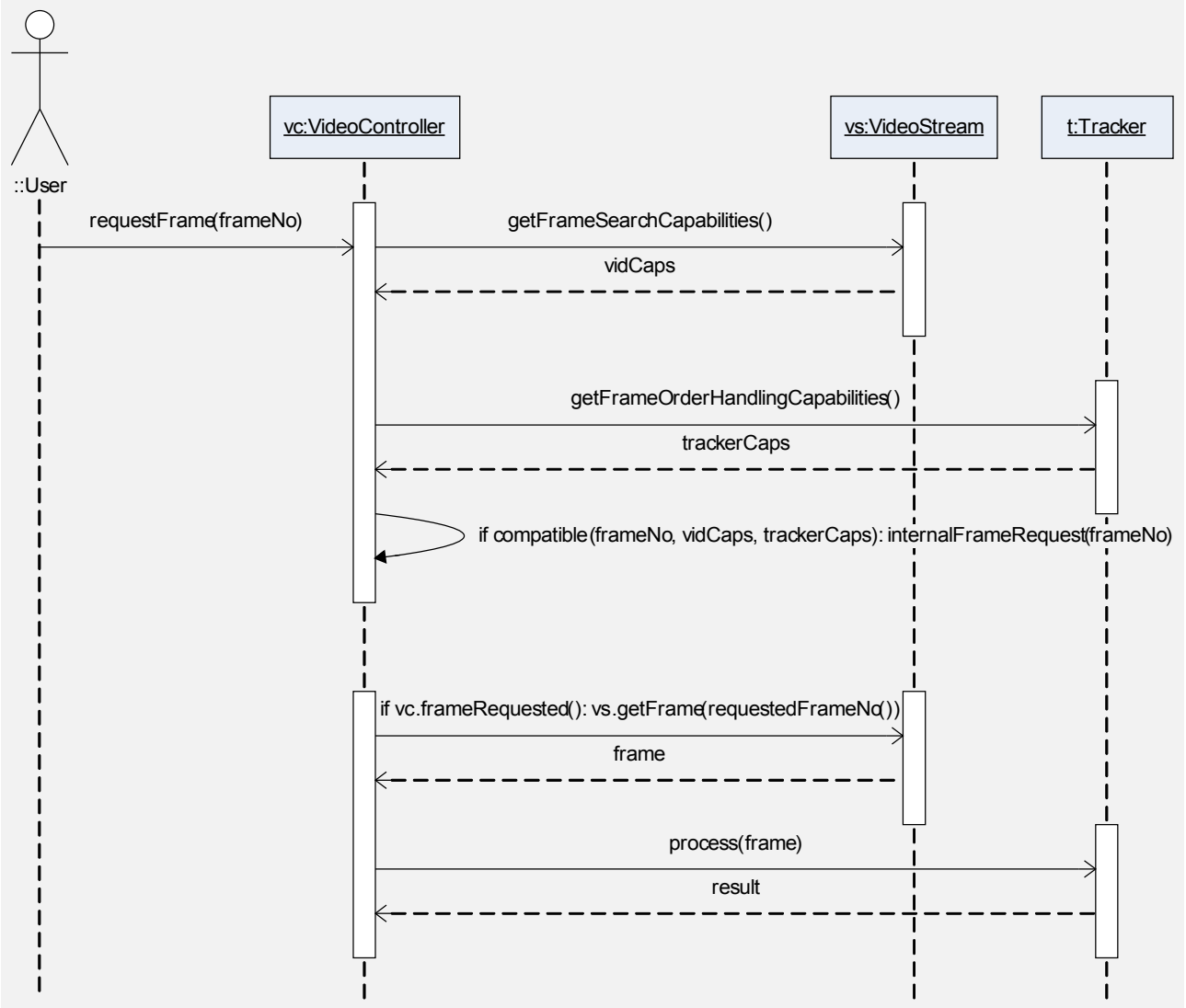
A similar problem arose when implementing the baton tracker interface, as tracking algorithms vary in their temporal capabilities. It is useful for testing purposes when a tracking algorithm allows us to retrack the current frame, or even a past frame, with modified parameters. Of course, we can always retrack the current frame or a past one if we are prepared to either (a) revert to a cached earlier state; (b) retrack the whole sequence again from the beginning; or (c) reinitialise the tracker at the required frame, causing it to discard all of the information it has gathered up to the current point in the video sequence. The third choice isn't generally a valid option however, as we would usually be interested in how the tracker would have performed if it had processed the video with the new parameters from the start of the sequence.

The first tracking algorithm we implemented, based on shape recognition (see section III.2.1), is temporally stateless in the sense that it gives the same output for any given frame regardless of when it occurred in the video stream, i.e. it supports random frame access[1]. The condensation algorithm on the other hand (see section III.2.2) can only retrack the current frame or a past one by one of the three methods listed in the previous paragraph. None of these methods are particularly desirable though as (a) caching the condensation tracker's state in every frame has a considerable memory requirement; (b) retracking the whole sequence again from the beginning would take too long; (c) reinitialisation would be invalid for the reason given above. To deal with this problem we again declared a method in the baton tracker interface that returns information about the tracker's ability to handle frames out of sequence.

The video streaming components and baton tracking components are controlled by a class within the thread labelled "video controller thread" in fig. III.1-1. We will refer to this class as the video controller. When the user requests a frame, the video controller first of all checks the frame searching capabilities of the video streaming component and the frame ordering requirements of the tracker component. If these are found to be compatible with the requested frame number, an internal request is made for the requested frame to be sent to the tracker on the next iteration of the video thread's loop. This process is shown in fig. III.1-3. Note that "requestFrame(frameNo)" is executed in the caller's thread space (which in this case would be the GUI thread space), whilst the section beginning "if(frameRequested())" is executed asynchronously in the video controller thread space.

1. This isn't strictly true, its estimation of the baton's velocity and acceleration vectors does depend on the time ordering of the frames, but the tracker doesn't use these vectors itself, it generates them for the benefit of the condensation algorithm-based tracker.

Fig. III.1-3 *– Sequence diagram showing how the video controller handles frame requests.*



III.2 Baton Tracking

Implementing two different tracking algorithms in this project allowed us to evaluate their performance relative to each other and assess their respective merits and shortcomings, particularly with respect to their suitability to real-time performance. The differing behaviour of the two algorithms made different demands on the beat detector, which will be discussed later.

NOTE: Any references made to pixel luminance within this section assume a normalised scale, where 1 represents full intensity and 0 represents zero intensity.

III.2.1 Shape Recognition-Based Tracking

In section II.3.2 we presented a metric $M(P)$ for determining how similar the shape of an image segment P is to that of a baton based on the length of P's central axis and the standard deviation of its pixels about its axis. We will now discuss the process by which we extracted the segments from each frame of the video stream. We will also explain how we calculated the central axis of each segment, which is prerequisite to the calculation of $M(P)$.

III.2.1.1 Background Subtraction

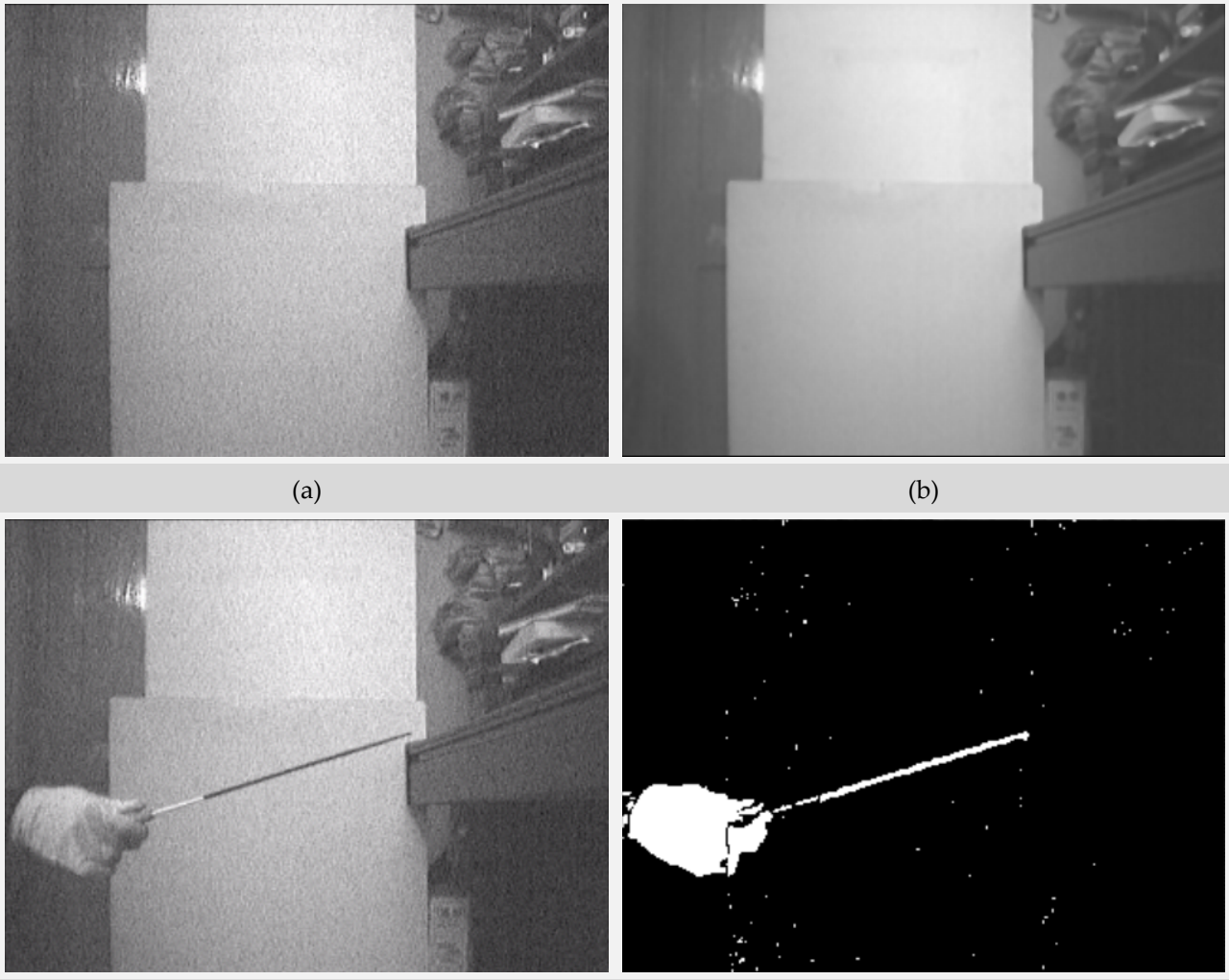
Fig. III.2-2 shows the main stages of our algorithm. Input images are generally very noisy, so to facilitate the segmentation phase we filtered each image with a median blur and used the filtered image to estimate an image of the background. Our background estimation procedure worked on the assumption that the background is static and that there are no significant changes in the lighting. These are fair assumptions to make considering that the system will typically only be used indoors.

We used the running average method to accumulate the background over a user-specified number of frames during which the scene is empty. Given a pixel p_t from the image that occurs at time t, this method approximates the average colour of the corresponding background pixel b_t as $b_t = (1\text{-}\alpha)b_{t-1} + \alpha \cdot p_t$, where $\alpha \in [0, 1]$ is a user-specified weighting factor.

Once the background image B has been calculated, the algorithm generates a mask M_B, which is initially used to separate any foreground objects in the current image I that have entered the scene (such as the user) from the background. The mask is calculated by comparing each pixel p_I of I to its corresponding pixel p_B in B, and masking out the pixel if $|p_I - p_B| < \tau$, where τ is a user-specified threshold.

The mask is used to indicate areas of the image that should be ignored when searching for candidate segments. Every time a new segment is found, the area of the mask that the segment covers is masked out so that the algorithm won't consider it again. Eventually every pixel of the image is masked out and the algorithm terminates, returning the segment P for which $M(P)$ gave the highest value or *NULL* if no segment was found for which the metric gave a value > 0.

Fig. III.2-1 *– Images from the background subtraction phase: (a) an input image of an empty scene; (b) the estimated background image averaged over 50 input images; (c) an input image featuring the user; (d) the background subtraction mask with the threshold set to 0.06 (black=background, white=foreground).*

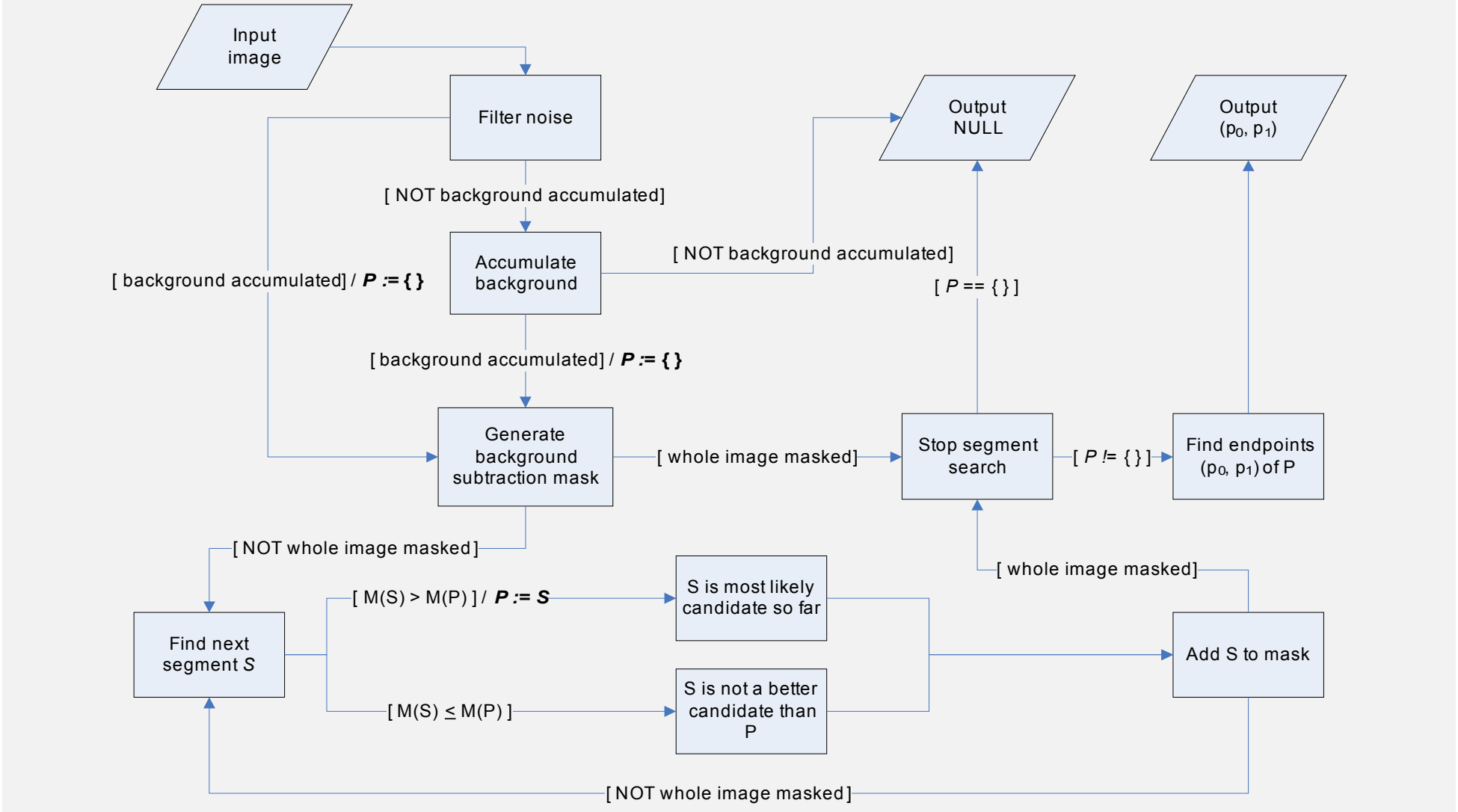


(a) (b)

(c) (d)

Fig. III.2-1 above shows an example of the results of this background subtraction technique. Fig. III.2-1 (b) shows how most of the noise from the initial source images is filtered out by averaging. In fig. III.2-1 (d) most of the background has been successfully removed using a relatively low threshold, although there are still a few noisy pixels that have evaded the filtering process. Notice also that some parts of the hand and baton have been mistakenly classified as part of the background. A higher threshold would remove the stray pixels that haven't been filtered out, although too high a threshold would cause further degradation to the hand and baton.

Fig. III.2-2 *– Flow chart showing the operation of the shape recognition-based tracker on a single image. Conditional transitions are indicated by square brackets. Actions to be taken when a condition holds follow a forward slash.*



III.2.1.2 Extracting Segments

Our segment extraction method was based on heuristics that relate to the following two laws from Gestalt psychology (taken from [WP4]):

> ***Law of Similarity*** *– Our minds group similar elements together. The similarity is determined by the form, colour, size and brightness of the elements.*
>
> ***Law of Proximity*** *– Elements that are close to each other spatially or chronologically are grouped together by our minds and seen as belonging together.*

Combining these laws gives us the heuristic that any two pixels that are of similar luminance and are in close proximity of each other are likely to be part of the same object. This is certainly true of the pixels that define the surface of the baton, as you can see in fig. III.2-1 (c).

We define two pixels as being close to one another if they are neighbours. To determine whether or not a pixel p is similar to its eight neighbours $N(p)$, our algorithm calculates the average absolute difference between p's intensity and the intensities of those neighbours. Again we used a threshold to define how small this average difference must be for p to be considered similar to its neighbours.

This forms the basis of the method we used to search for the pixels that make up each segment. The algorithm chooses an initial unmasked pixel p and tests whether or not it is similar to its neighbours $N(p)$. If they are similar, the algorithm then tests each pixel p' of $N(p)$ to see whether or not it is similar to *its* neighbours $N(p')$, and so on until no more similar neighbouring pixels are found. All of the pixels that satisfy the similarity test are added to a set P that defines the segment.

Put more formally, the algorithm constructs an image segment P located about a point p with intensity *f(p)* as defined by the following recursive function:

$$P = h(p,\tau) = \begin{cases} \emptyset, & masked(p) \vee p \in P \\ \{p\}, & \neg masked(p) \wedge \neg s(p,\tau) \\ \{p\} \underset{p' \in N(p)}{\cup} h(p',\tau), & \neg masked(p) \wedge s(p,\tau) \wedge p \notin P \end{cases}$$

$$s(p,\tau) \longleftrightarrow \sum_{p' \in N(p)} \frac{|f(p) - f(p')|}{|N(p)|} < \tau$$

Where:

τ is the similarity threshold.

Of course, using this recursive function to compute P would be inefficient. Instead, we adapted the scanline-based region-filling algorithm given in [KF1]. This algorithm was appropriate to use because it fills regions by iterating across neighbouring pixels starting from a seed point. To adapt it to the task of segment extraction, we used our similarity test

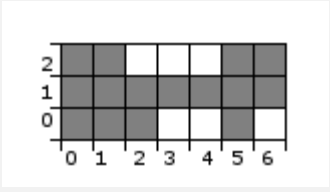
to indicate the boundary of the region and we added each new pixel that the region filler found to a data structure that represents the image segment.

III.2.1.3 An Image Segment Data Structure

The nature of the segmentation algorithm necessitates the use of a dynamic data structure to represent the segments. The simplest data structure that we could have used to do this would have been a stack of 2D coordinates. This representation isn't without problems though.

Suppose each node of the stack requires 12 bytes (8 bytes for the coordinates + 4 bytes for the pointer to the next node). In a typical video sequence with a frame size of 320*240 (the size we used in our tests), the tracker would need to dynamically allocate and deallocate memory for tens of thousands of stack nodes per frame[1], in addition to the data structures used by the region filler[2]. On our test machine we found this to cause significant performance problems, possibly due to memory fragmentation, as the order of the allocations and deallocations was highly dependent on the input image.

Fig. III.2-3 *– An example image segment.*



To solve this problem, we developed a more memory-efficient data structure that operates in a manner analogous to run-length-encoding-based data compression. The data structure is a list of sorted trees of pixel spans. A tree in this data structure contains all of the pixel spans that lie on a single scanline. The list contains a single tree for each scanline that the segment covers.

A span of pixels from point (x_0, y) to (x_1, y) is represented as (x_0, x_1+1). Thus, the shaded segment shown in fig. III.2-3 would be represented as a list L of three trees as follows:

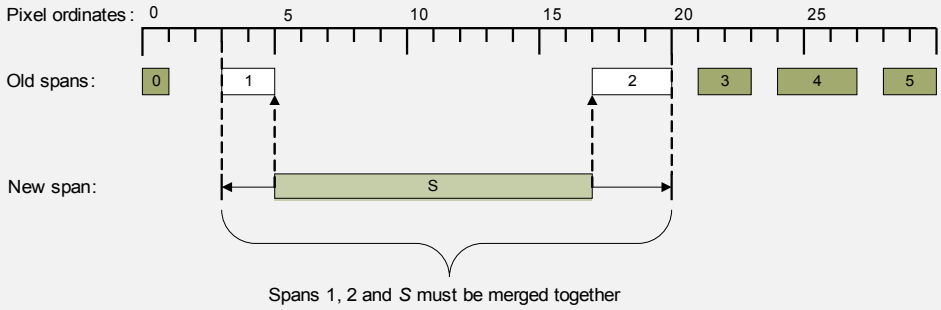
```
L[2] = Tree{(0, 2), (5, 7)}
L[1] = Tree{(0,7)}
L[0] = Tree{(0, 3), (5, 6)}
```

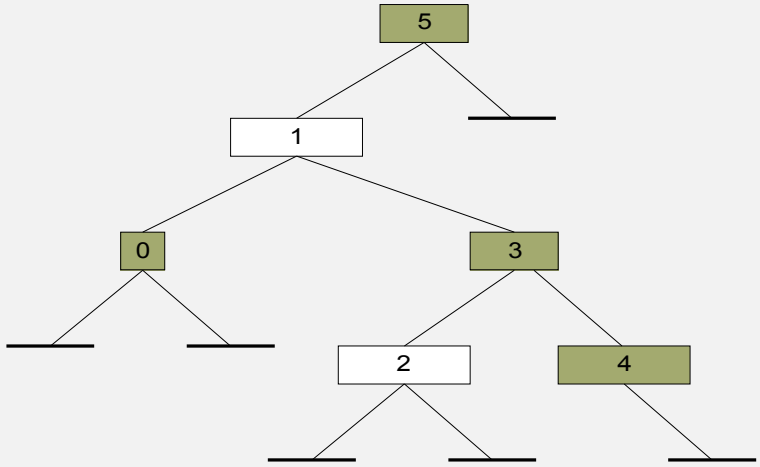
Each node of the tree contains a single pixel span and points to two child nodes. The spans are sorted in each tree according to their position on the scanline. The tree nodes are ordered in the usual way – the left subtree of node t contains all of the spans that are < t's span, and the right subtree of t contains all of the spans that are > t's span. To maintain this ordering when adding a new span s to the tree, s is merged with any spans currently in the tree that it lies adjacent to. The most general case that arises is illustrated in fig. III.2-4.

1. In the worst case when the segments cover the entire frame, 320*240=76800 nodes would be needed.
2. The region filler uses a stack of 20-byte elements representing spans of contiguous pixels on single scanlines. New spans are pushed onto the stack every time a segment boundary on a scanline is reached.

Fig. III.2-4 *– Adding a new pixel span to a scanline tree.*

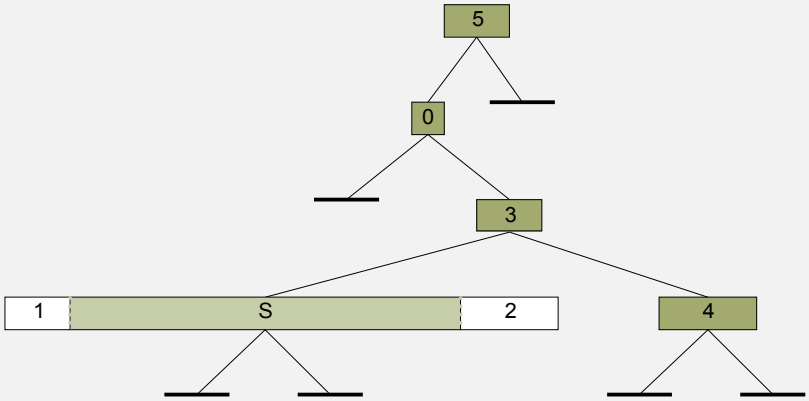
(Right) Spans 0-5 are spans that the region filler has found so far. Span *S* is a new span it has found that is to be added to the tree. It must be merged with spans 1 & 2 and inserted in their place. After *S* is merged, it will cover pixels 3–19.





(Left) The pixel span tree before *S* is merged and added. There are of course many possible valid sorted arrangements for the nodes of this tree.

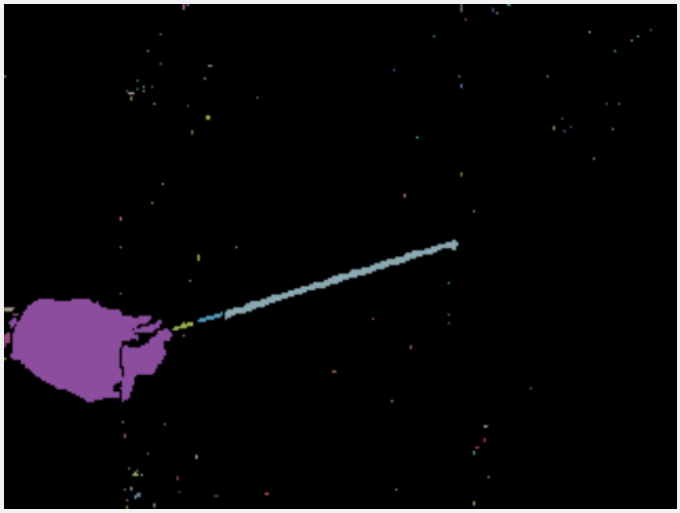
(Right) The pixel span tree after removing spans 1 & 2, merging them with *S* and adding the new merged span to the tree.



Note that the theoretical possibility of a span being added to the tree that covers spans already contained within the tree never arises in practise, as the region filler never iterates over any given span more than once.

The toy example given in fig. III.2-3 wouldn't benefit greatly from the use of this data structure. A more realistic example however, such as the hand in fig. III.2-5, would require much less memory under this representation than it would if it was represented as a stack of pixel coordinates. In the worst-case example of an image segment that covers an entire image of size $w*h$, this data structure would require a list of h trees, each of which contains a single span, so the memory requirement would be O(h). In contrast the memory requirement for a stack of pixels used to represent such a segment would be O($w*h$).

Fig. III.2-5 *– Image of the extracted segments after segmentation (segments are assigned distinct colours).*



III.2.1.4 Calculating the Central Axis

After extracting each segment P from the image, the next stage in the algorithm is to calculate the segment's central axis and the standard deviation of the perpendicular distances from each pixel to this axis. To do this we used theorem II.3.2.2-1.

On first sight, the closed-form solution given in theorem II.3.2.2-1 seems to suggest multiple passes over the pixel coordinate data to calculate the point $\bar{p}$ on the axis and the axis' directional vector $(\cos\theta, \sin\theta)$. In fact both of these parameters can be calculated in a single pass:

Given N pixel coordinates $p_0, p_1, \ldots, p_{N-1}$ with mean $\bar{p} = (\bar{p}_x, \bar{p}_y)$, we can calculate the terms A and B as follows:

$$
\begin{aligned}
A &= \frac{1}{2}\sum (v_{iy}^2 - v_{ix}^2) \\
&= \frac{1}{2}\sum ((p_{iy} - \bar{p}_y)^2 - (p_{ix} - \bar{p}_x)^2) \\
&= \frac{1}{2}\sum (p_{iy}^2 - 2\,\bar{p}_y\,p_{iy} + \bar{p}_y^2 - (p_{ix}^2 - 2\,\bar{p}_x\,p_{ix} + \bar{p}_x^2)) \\
&= \frac{1}{2}\Big[\sum p_{iy}^2 - 2\,\bar{p}_y \sum p_{iy} + N\,\bar{p}_y^2 - (\sum p_{ix}^2 - 2\,\bar{p}_x \sum p_{ix} + N\,\bar{p}_x^2)\Big] \\
&= \frac{1}{2}\Big[\sum p_{iy}^2 - 2N\,\bar{p}_y^2 + N\,\bar{p}_y^2 - (\sum p_{ix}^2 - 2N\,\bar{p}_x^2 + N\,\bar{p}_x^2)\Big] \\
&= \frac{1}{2}\Big[\sum p_{iy}^2 - \sum p_{ix}^2 + N(\bar{p}_x^2 - \bar{p}_y^2)\Big] \\
&= \frac{1}{2}\left[\sum p_{iy}^2 - \sum p_{ix}^2 + \frac{1}{N}\left\{\left(\sum p_{ix}\right)^2 - \left(\sum p_{iy}\right)^2\right\}\right]
\end{aligned}
$$

$$\begin{aligned} B &= \sum v_{ix} v_{iy} \\ &= \sum (p_{ix} - \bar{p}_x)(p_{iy} - \bar{p}_y) \\ &= \sum \left(p_{ix} p_{iy} - \bar{p}_y p_{ix} - \bar{p}_x p_{iy} + \bar{p}_x \bar{p}_y\right) \\ &= \sum p_{ix} p_{iy} - N \bar{p}_x \bar{p}_y \\ &= \sum p_{ix} p_{iy} - \frac{1}{N}\left[\sum p_{ix}\right]\left[\sum p_{iy}\right] \end{aligned}$$

Where:

$$v_i = p_i - \bar{p}$$

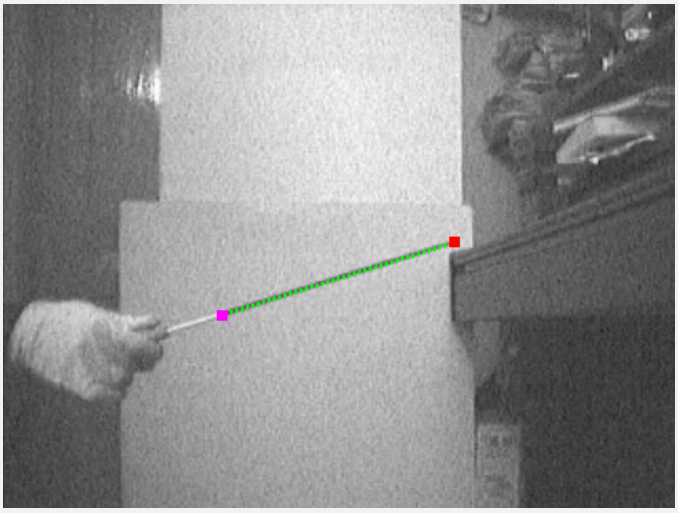
These sums can all be calculated in a single pass, allowing us to calculate $\bar{p}$ *and the set* Ω *defined in the theorem. We can then evaluate* $R(\omega)$ *for any* $\omega = (\omega_x, \omega_y) \in \Omega$ *as follows:*

$$\begin{aligned} R(\omega) &= \sum \left[v_i \cdot v_i - (v_i \cdot \omega)^2\right] \\ &= \sum v_i \cdot v_i - \sum (v_{ix}\omega_x + v_{iy}\omega_y)^2 \\ &= \sum v_i \cdot v_i - \left[\omega_x^2 \sum (p_{ix} - \bar{p}_x)^2 + 2\omega_x\omega_y \sum (p_{ix} - \bar{p}_x)(p_{iy} - \bar{p}_y) + \omega_y^2 \sum (p_{iy} - \bar{p}_y)^2\right] \\ &= \sum v_i \cdot v_i - \left[\omega_x^2 \left\{\sum p_{ix}^2 - N\,\bar{p}_x^2\right\} + 2\,\omega_x\omega_y \left\{\sum p_{ix}\,p_{iy} - N\,\bar{p}_x\,\bar{p}_y\right\} + \omega_y^2 \left\{\sum p_{iy}^2 - N\,\bar{p}_y^2\right\}\right] \end{aligned}$$

The only one of these sums whose value we don't already know from our calculation of A and B is $\sum v_i \cdot v_i$ *. This term is constant however when we are searching for the argument* $\omega \in \Omega$ *that minimises R, and so it needn't be calculated at all.*

Once we've found the parameters of the central axis, we then search for the end points of the segment by transforming the coordinates of its pixels as described at the end of section II.3.2.2. This requires a second pass over the coordinate data.

Fig. III.2-6 *– Image of the tracked baton points (shown as coloured squares) and the baton's central axis (shown as a green dotted line).*

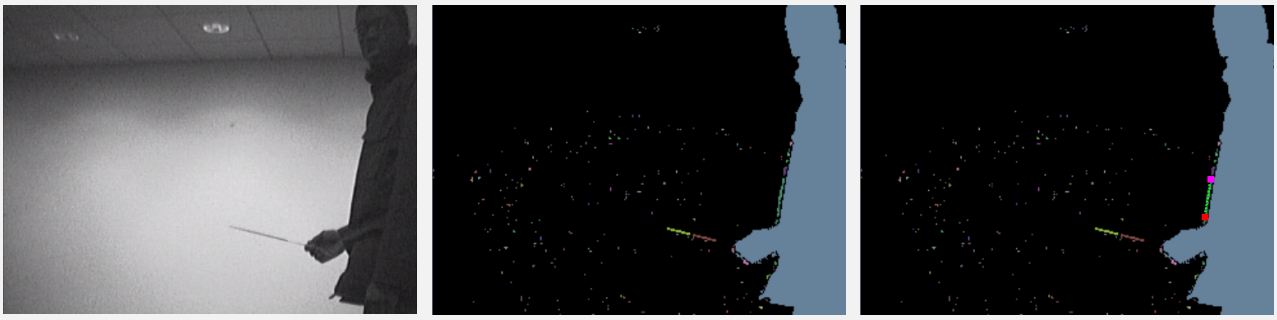


At this point we can finally calculate the value of the metric $M(P)$. Fig. III.2-6 shows the result of the algorithm on the input image presented in fig. III.2-1 (c). The algorithm has correctly identified the black part of the baton in this case (the baton was painted black to make it stand out against the white background).

III.2.1.5 Conclusion

From what we have discussed so far, the tracker can give the end points of any segment P in the image that maximises the value of the metric $M(P)$. However this doesn't guarantee that the points it outputs are true positives. An example of a false match is shown in fig. III.2-7.

Fig.III.2-7 *– These images show how the tracker can generate a false positive. In image (b), the centre of the baton has been mistakenly classified as being part of the background, and so the baton is split into two unconnected segments. There also just happens to be a thin segment near the user's body that's longer than the two segments of the baton, and so this longer segment is incorrectly identified as the baton, as shown in image (c).*



(a) (b) (c)

This figure identifies the two main causes of tracker failure:

1. The tracker may fail to locate the baton if the background subtraction stage splits the baton into multiple segments.
2. The tracker will fail to locate the baton if there is another segment in the image that's longer than the baton segment and thin enough to not be discarded.

The first cause of failure in this list could have been avoided by joining up collinear segments that are within a short enough distance of one another to support the possibility that they are part of the same object. Due to time constraints however, we were unable to do this.

The second cause of failure can be reduced by taking the expected motion of the baton into account. The baton generally moves short distances over consecutive frames, and its motion follows a cyclic path, as shown in fig. II.2-2. Thus, by incorporating information about the motion of the baton into a tracking algorithm, we can reduce the degree to which the path it outputs deviates from the true path of the baton due to background clutter. This was our motivation for using the condensation algorithm, and our approach to its implementation is discussed in section III.2.2.

The tracker will also fail if the background subtraction threshold is set so low that the tracker fails to identify the region of the image surrounding the baton as being part of the background, or if the threshold is set so high that the tracker cannot distinguish the baton from the background. However these two causes of failure can generally be avoided by setting the threshold to a value somewhere in-between.

III.2.2 Tracking with the Condensation Algorithm

In section II.3.3 we presented the theoretical basis of the condensation algorithm. We did not, however, discuss: how to sample from and evaluate the motion model $P(X_t|X_{t-1})$, how to evaluate the likelihood distribution $P(Z_t|X_t)$ or how to sample from and evaluate the importance function $g(X_t)$. The nature of these distributions is problem-specific and will be discussed in the following sub-sections.

III.2.2.1 Sampling $P(X_t|X_{t-1})$

Recall from section II.3.3.2 that sampling from the motion model is the second step in sampling from the prior $P(X_t \mid \mathcal{Z}_{t-1})$. We would ideally like samples drawn from this distribution to be as close as possible to the true position of the baton in frame t, given the information we have about the position and motion of the baton in frame t-1.

Let T_t be the time in seconds at which frame t occurs. The duration, δ_{t-1} of frame t-1 is then $\delta_{t-1} = T_t - T_{t-1}$. If δ_{t-1} is small for any given t, we can reasonably assume that the acceleration of the baton's end points at time T_t will be similar to their acceleration at time T_{t-1}. This gives us the basis for the deterministic part of our motion model.

We define the state X_t of the baton in frame t as follows:

$$X_t \equiv (P_{0,t}, V_{0,t}, A_{0,t}, P_{1,t}, V_{1,t}, A_{1,t})$$

Where:

($P_{0,t}$, $P_{1,t}$) = the positions of the baton's end points at time T_t.

($V_{0,t}$, $V_{1,t}$) = the velocity of each of the baton's end points at time T_t.

($A_{0,t}$, $A_{1,t}$) = the acceleration of each of the baton's end points at time T_t.

The assignment of the indices 0 and 1 to each of the baton's end points is arbitrary, although it should of course remain consistent from frame to frame.

Given X_t and $P_{i,t}$ for each $i \in \{0,1\}$, we can estimate the other derivatives of its motion as:

$$V_{i,t} \approx \frac{P_{i,t} - P_{i,t-1}}{\delta_{t-1}}, \quad A_{i,t} \approx \frac{V_{i,t} - V_{i,t-1}}{\delta_{t-1}}$$

These estimates tend to the true values of the derivatives as $\delta_{t-1} \longrightarrow 0$, and so given a high enough frame rate we could potentially estimate even more derivatives in the same manner. In practise though, the maximum achievable frame rate is limited by the camera and the efficiency of the algorithm. Besides, having an estimate for the acceleration vectors is good enough to allow our algorithm to predict curved trajectories.

Before calculating these derivatives, a value for $P_{i,t}$ needs to be obtained. We used our assumption that $A_{i,t} \approx A_{i,t-1}$ for all i, t to predict $P_{i,t}$ using the formula $s = ut + \frac{1}{2}at^2$, which gives the distance s travelled by an object in terms of its initial velocity, u, its acceleration, a, and the duration of its motion, t. Our predictive formula is:

$$P_{i,t} = P_{i,t-1} + V_{i,t-1}\delta_{t-1} + \frac{1}{2}A_{i,t-1}\delta_{t-1}^2 + \Omega\,\delta_{t-1} \qquad \text{(Eq. III.2.2.1)}$$

Where:

Ω is a 2D random variable that follows a bivariate Gaussian distribution.

We include Ω to allow for sudden changes in a baton point's velocity vector. So to sample from $P(X_t | X_{t-1})$, we calculate the deterministic part of eq. III.2.2.1 and we add to it a sample drawn from the distribution of Ω scaled by δ_{t-1}. We assume that the mean of Ω is zero, as the deterministic part of the equation is the best guess we can make about the baton's position at time t from the information we have.

Thus we used a 4D Gaussian distribution to model unpredictable changes in the x and y ordinates of the velocity of the baton's base and tip. The covariance matrix of this distribution represents information about the relationship between these velocity variables. We would expect there to be high covariance between the y ordinates of the baton's base and tip, and so a good estimate of the covariance matrix at each time t should lead to more accurate predictions. Due to a lack of time however we were unable to do this, so we assumed zero covariance between these variables, which represents the tracker having no information about the relationship between these variables. This still produced acceptable results given a large enough sample set size, as shall be discussed in section IV.2.

III.2.2.2 Evaluating $P(Z_t | X_t)$

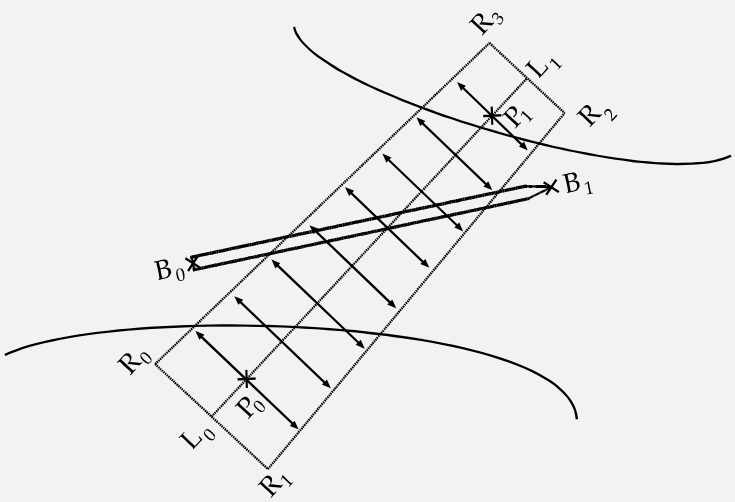
When the baton can be distinguished from its background (this is an implicit assumption for both of our trackers), an edge-detecting convolution filter generally gives high edge strength values at the baton's boundary. This observation makes the set of extracted edges a suitable candidate for our Z_t data.

The value of $P(Z_t | X_t)$ can be interpreted as a measure of the likelihood of frame t having the edges that it has given that the baton is in state X_t. Notice the use of the word "likelihood". $P(Z_t | X_t)$ is only ever evaluated as part of eq. II.3.3.2 where it is normalised over the samples of S_t. Hence we needn't evaluate $P(Z_t | X_t)$ as a normalised probability here, we can calculate it as an unnormalised likelihood value. To avoid confusion, we shall now refer to this likelihood as $L(Z_t | X_t)$, or L_t for short.

The effect of L_t should be to increase the chance of strong hypotheses in the sample set S_t being used as the basis for predicting the samples of S_{t+1} and to reduce the chance of

weak hypotheses being used for prediction. A perfect hypothesis, $\varsigma_t = (p_{0,t}, v_{0,t}, a_{0,t}, p_{1,t}, v_{1,t}, a_{1,t})$ would be one that gives the exact position of the baton's end points in frame t. In terms of Z_t's data, we would expect to see 2 strong edges (the baton's boundary) on either side of and parallel to the line $p_{1,t} - p_{0,t}$. L_t should be at its peak value in this case to ensure that ς_t is more likely to be sampled from S_t in the next iteration of the algorithm than any other weaker hypothesis.

Fig. III.2-8 *– The general case in which we need to evaluate L_t.*



Of course there's no guarantee of a perfect hypothesis being generated, so in general we want L_t to act as a hypothesis metric whose value is high when the two hypothesised end points are near a pair of near-parallel edges. Thus we need to search for the baton's edges in the area around each hypothesis.

Searching the entire image for each of the N samples would be highly inefficient. It would be pointless in fact, because the value of L_t should be low when the hypothesised points are far from the true position of the baton, not to mention the fact that a search of the whole image would almost certainly cause our evaluation of L_t to be thrown off by clutter.

The general case is illustrated in fig. III.2-8. The labels indicate the following:

- B_0 and B_1 are the true baton end points. The dark tube around them represents the baton's edges. This tube may have gaps in it due to noise in the source image.
- P_0 and P_1 are the hypothesised baton end points.
- Quadrilateral $R_0R_1R_2R_3$ is the search window. $|L_0R_0| = |L_0R_1| = |P_0L_0|$ and $|L_1R_3| = |L_1R_2| = |P_1L_1|$. $|P_0L_0|$ and $|P_1L_1|$ can be set according to our belief in the accuracy with which P_0 and P_1 were predicted.
- Line L_0L_1, which passes through P_0 and P_1, is the central axis of $R_0R_1R_2R_3$, from which the search should be directed.
- The arrows are the actual lines about L_0L_1 that we will search along. Again, for the sake of efficiency, it would be better to sample the search window rather than consider every pixel of it.
- The other solid lines are examples of background clutter that could throw off the search.

We will now define how we used the information about the edges that the search lines (i.e. the arrows in the above figure) intersect to determine a suitable value for L_t.

Let Λ be the set of search lines on one side of L_0L_1, and let Λ' be the set of search lines on the other side ($\Lambda''=\Lambda$ and $|\Lambda|=|\Lambda'|$). Furthermore, let $\lambda \in \Lambda$ and $\lambda' \in \Lambda'$ be adjacent search lines on opposite sides of L_0L_1 (again, $\lambda''=\lambda$).

Define: $\Lambda_{[i]}$ as the i^{th} search line of Λ, enumerated according to the order in which they occur when moving from L_0 to L_1; $N(\lambda)$ as the number of edges λ intersects; and $\lambda_{[i]}$ as the distance between L_0L_1 and the i^{th} edge that λ intersects such that $\forall i, j\,(0 \le i < j < N(\lambda) \rightarrow \lambda_{[i]} < \lambda_{[j]})$.

We can then define the following statistics based on these sets for any $\tilde{\Lambda} \in \{\Lambda, \Lambda'\}$:

$$\mu_+(\tilde{\Lambda}) = \begin{cases} \frac{1}{2} E\left[|\lambda_{i[0]} - \lambda'_{i[0]}|\right] & for\ \lambda_i \in \tilde{\Lambda}_{\mu+} \quad , |\tilde{\Lambda}| > 0 \\ k_1 & , otherwise \end{cases}$$

$$\sigma_+^2(\tilde{\Lambda}) = \begin{cases} E\left[(\lambda_{i[0]} + \lambda'_{i[0]})^2\right] - E\left[\lambda_{i[0]} + \lambda'_{i[0]}\right]^2 & for\ \lambda_i \in \tilde{\Lambda}_{\mu+} \quad , |\tilde{\Lambda}_{\mu+}| > 1 \\ k_2 & , otherwise \end{cases}$$

Where:

$$\tilde{\Lambda}_{\mu+} = \{\lambda;\ \lambda \in \tilde{\Lambda},\ N(\lambda) > 0,\ N(\lambda') > 0\}$$

$$\mu_- = \begin{cases} \frac{1}{2} E\left[\lambda_{i[1]} - \lambda_{i[0]}\right] & for\ \lambda_i \in \tilde{\Lambda}_{\mu-} \quad , |\Lambda_{\mu-}| > 0 \\ k_3 & , otherwise \end{cases}$$

$$\sigma_-^2(\tilde{\Lambda}) = \begin{cases} E\left[(\lambda_{i[1]} - \lambda_{i[0]})^2\right] - 4\,\mu_-^2(\tilde{\Lambda}_{\mu-}) & , |\tilde{\Lambda}_{\mu-}| > 1 \\ k_4 & , otherwise \end{cases}$$

Where:

$$\tilde{\Lambda}_{\mu-} = \{\lambda;\ \lambda \in \tilde{\Lambda},\ N(\lambda) > 1\}$$

$$\sigma_\delta^2(\tilde{\Lambda}) = \begin{cases} E\left[\left(\frac{(\tilde{\Lambda}_{[k]})_{[0]} - (\tilde{\Lambda}_{[j]})_{[0]}}{k-j}\right)^2\right] - E\left[\frac{(\tilde{\Lambda}_{[k]})_{[0]} - (\tilde{\Lambda}_{[j]})_{[0]}}{k-j}\right]^2 & for\ (j,k) \in \tilde{\Lambda}_\delta \quad , |\tilde{\Lambda}_\delta| > 1 \\ k_5 & , otherwise \end{cases}$$

Where:

$$\tilde{\Lambda} = \{(j,k);\ 0 \le j < k < |\tilde{\Lambda}|, N(\tilde{\Lambda}_{[j]}) > 0, N(\tilde{\Lambda}_{[k]}) > 0, \neg \exists x (j < x < k,\ N(\tilde{\Lambda}_{[x]}) > 0)\}$$

$\mu_-(\tilde{\Lambda})$ and $\mu_+(\tilde{\Lambda})$ are the mean distances between L_0L_1 and the midpoints of the pairs of intersections that are closest to it. $\mu_-(\tilde{\Lambda})$ is for the case when both of the baton's edges lie on one side of L_0L_1, and $\mu_+(\tilde{\Lambda})$ is for the case when the edges lie on either side of it. They measure how close the central axis of the closest pair of edges to L_0L_1 is, evaluating to 0 when the central axis lies on L_0L_1.

$\sigma^2_-(\tilde{\Lambda})$ and $\sigma^2_+(\tilde{\Lambda})$ give the variances of the distances between the pairs of intersections that are closest to L_0L_1. As before, $\sigma^2_-(\tilde{\Lambda})$ is for the case when the baton's edges lie on one side of L_0L_1, and $\sigma^2_+(\tilde{\Lambda})$ is for the case when the edges lie on either side of it. They measure how parallel the closest edges to L_0L_1 are, evaluating to 0 when they're exactly parallel.

$\sigma^2_\delta(\tilde{\Lambda})$ is the variance of the differences between the lengths of the closest intersections of adjacent search lines to L_0L_1. It measures how straight the closest edge to L_0L_1 is, evaluating to 0 when it's perfectly straight.

$k_1, k_2, \ldots, k_5$ are default constants for when these statistics are undefined.

When all of these statistics are low, they define properties that we would expect of Z_t given a hypothesis X_t that is close to the true state of the baton, i.e. they represent the fact that there should be a pair of straight, parallel edges near X_t. The greater any of these statistics gets, the less likely it is that the edges near X_t are those of the baton. Given this, we combined them to define L_t as follows:

$$L_t = 1+\left(1+\frac{|\Lambda_{\mu-}|}{1+\mu^2_-(\Lambda)}+\frac{|\Lambda'_{\mu-}|}{1+\mu^2_-(\Lambda')}+\frac{|\Lambda_{\mu+}|}{1+\mu^2_+(\Lambda)}\right) \times\left(1+\frac{|\Lambda_{\mu-}|}{1+\sigma^2_-(\Lambda)}+\frac{|\Lambda'_{\mu-}|}{1+\sigma^2_-(\Lambda')}+\frac{|\Lambda_{\mu+}|}{1+\sigma^2_+(\Lambda)}\right) \times\left(1+\frac{|\Lambda_{\delta}|}{1+\sigma^2_\delta(\Lambda)}+\frac{|\Lambda'_{\delta}|}{1+\sigma^2_\delta(\Lambda')}\right) \quad \text{(Eq. III.2.2.2)}$$

Note that $\mu^2_+(\Lambda)=\mu^2_+(\Lambda')$ and $\sigma^2_+(\Lambda)=\sigma^2_+(\Lambda')$, and so only one of each of these terms is included. By scaling each term by the cardinality of the relevant subset of Λ or Λ', we give more weight to terms that have more data to support them.

III.2.2.3 The Importance Function and (Re)initialisation

We have now discussed all of the main aspects of our implementation of the condensation algorithm except for (re)initialisation. As noted in section II.3.3.3, we can conveniently take the importance function $g(X_t)$ as the prior distribution at time t and use it to initialise the system.

We used our shape recognition-based tracker to define $g(X_t)$. Recall that the shape recognition algorithm discards segments with a non-zero likelihood in favour of the seg-

ment that has the highest metric value. Rather than discarding these segments, we took them as the range of $g(X_t)$, and took their normalised metric values as their probabilities. The motivation behind doing this is that by including the smaller segments in the distribution we can include any incomplete segments of the baton (such as the two unconnected segments shown in fig. III.2-7), making it possible for the algorithm to draw some samples that are close to the baton's true position. When these incomplete segments are very small however, their normalised probabilities become very small, and so they do not have much influence on the tracker. As mentioned before, the best way to fix this problem would have been to join up all collinear segments that are close to each other.

Having defined the importance function, we also need to define the criteria for automatic reinitialisation. To do this, we considered the hypotheses of sample set S_t as if they were members of a committee.

When there is a consensus amongst the members of a committee, it is generally assumed that the thing they have agreed upon is the best course of action to take (although it is quite possible that they are all wrong!). Disagreement occurs when various factors influence the committee members in different directions. To express this in terms of our sample set, the degree to which the samples disagree can be measured by how spread out they are. A "consensus" occurs when they all coincide with one another.

We used the following formula to measure the degree of disagreement $D(S_t)$ of the samples:

$$D(S_t) = \begin{cases} \dfrac{\sigma}{\sigma_M} & , \sigma \le \sigma_M \\ 1 & , otherwise \end{cases}$$

$$\sigma = \sqrt{\sigma_{0,x}^2 + \sigma_{0,y}^2 + \sigma_{1,x}^2 + \sigma_{1,y}^2}$$

Where:

$\sigma_{0,x}^2$, $\sigma_{0,y}^2$, $\sigma_{1,x}^2$, $\sigma_{1,y}^2$ are the variances of the x and y ordinates of the tips and bases of the samples.

σ_M is a user-defined threshold.

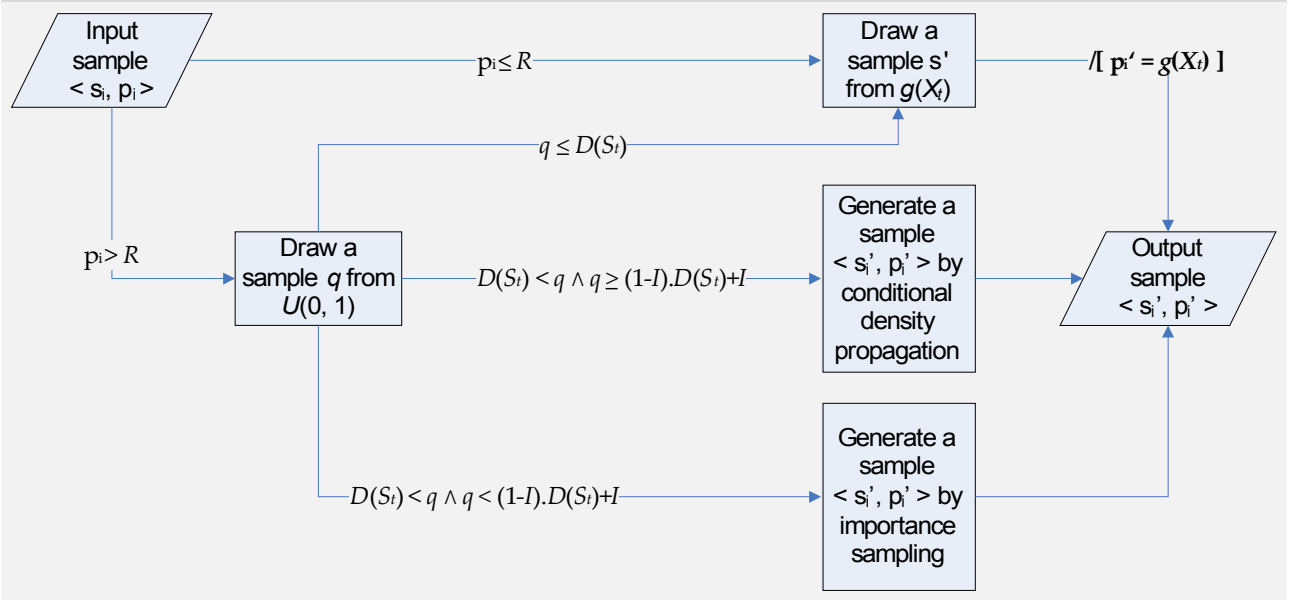
σ can be thought of as a generalisation of the definition of standard deviation to four dimensions. It equates to zero when the hypotheses all coincide exactly, otherwise it gives some greater positive value. σ_M defines a limiting acceptable value for σ, which allows us to take $D(S_t)$ as the probability of reinitialising any given sample. When σ exceeds this limit, $D(S_t) = 1$, and so all of the samples are reinitialised.

In addition to this, we used a rejection threshold R, which defines the minimum acceptable probability for any sample. Any sample whose probability is smaller than this threshold is reinitialised unconditionally. This is to eliminate the possibility of highly unlikely hypotheses remaining in the sample set for too long. Finally, we defined another

parameter I that gives the probability of generating a new sample from a sample s by importance sampling rather than by conditional density propagation, given that s is not going to be reinitialised.

Fig. III.2-9 shows our scheme for generating new samples. The figure doesn't show how the scheme fails (and outputs NULL) if $g(X_t)$ is empty when a rejected sample needs to be reinitialised from it.

Fig. III.2-9 *– Flowchart showing our sample generation scheme for the condensation algorithm.*



III.3 Gesture Interpretation

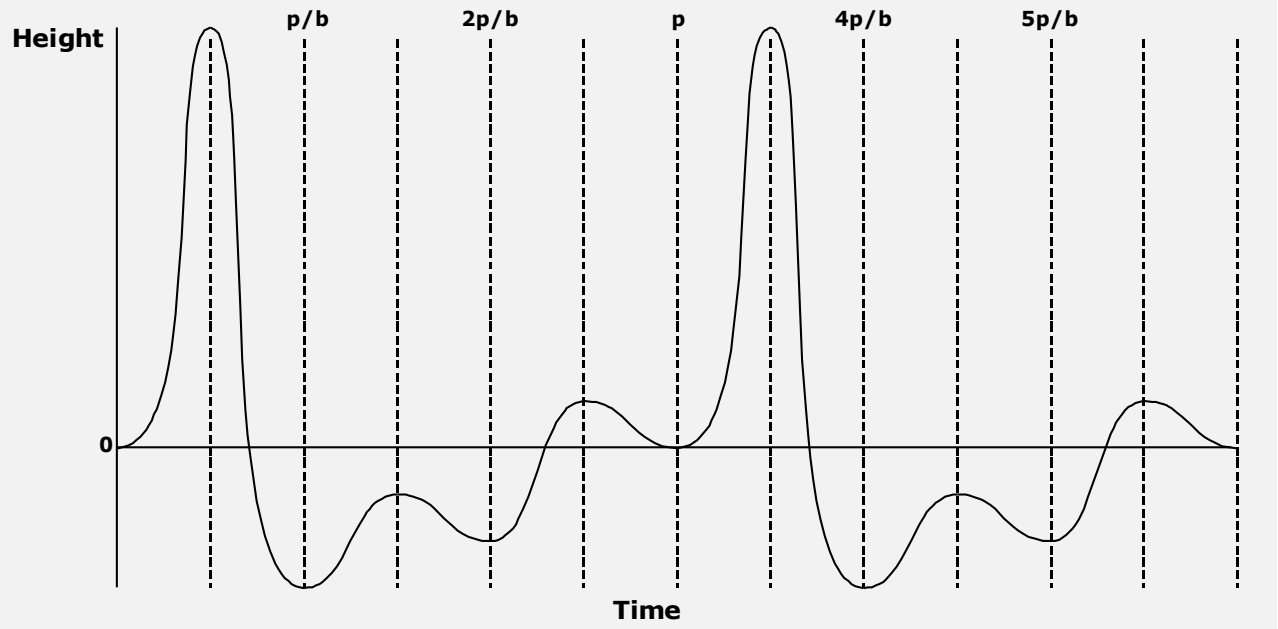
Whilst it should be clear by now that tracking was the main focus of this project, tracking alone wasn't enough for us to realise our goal. The tracked trajectories of the baton's endpoints need to be subjected to further analysis so that the conductor's tempo can be determined, and so that the time of the next half beat can be predicted.

III.3.1 Beat Detection

The trajectories output by the tracker are temporal discretisations of the true periodical motion of the baton. Fig. III.3-1 is an idealised graph of this motion. The period of the motion is p, and there are b=3 beats per bar. The dotted lines, which occur with a frequency $2b/p$ Hz, indicate when the half beats occur. From Nyquist's theory, we know that to be able to sample a periodical function, our sampling rate must be at least twice the frequency of that function.

So suppose the conductor is conducting at B beats/min = $B/30$ half beats/sec. The lower bound on the sampling rate we must achieve in order to detect the half beats is $B/15$ Hz. From our discussions with conductors, we found that the maximum tempo that a conductor can comfortably conduct at is approx. 180 beats/min. So by Nyquist's theory our system wouldn't need to run any faster than 180/15 = 12 frames/sec.

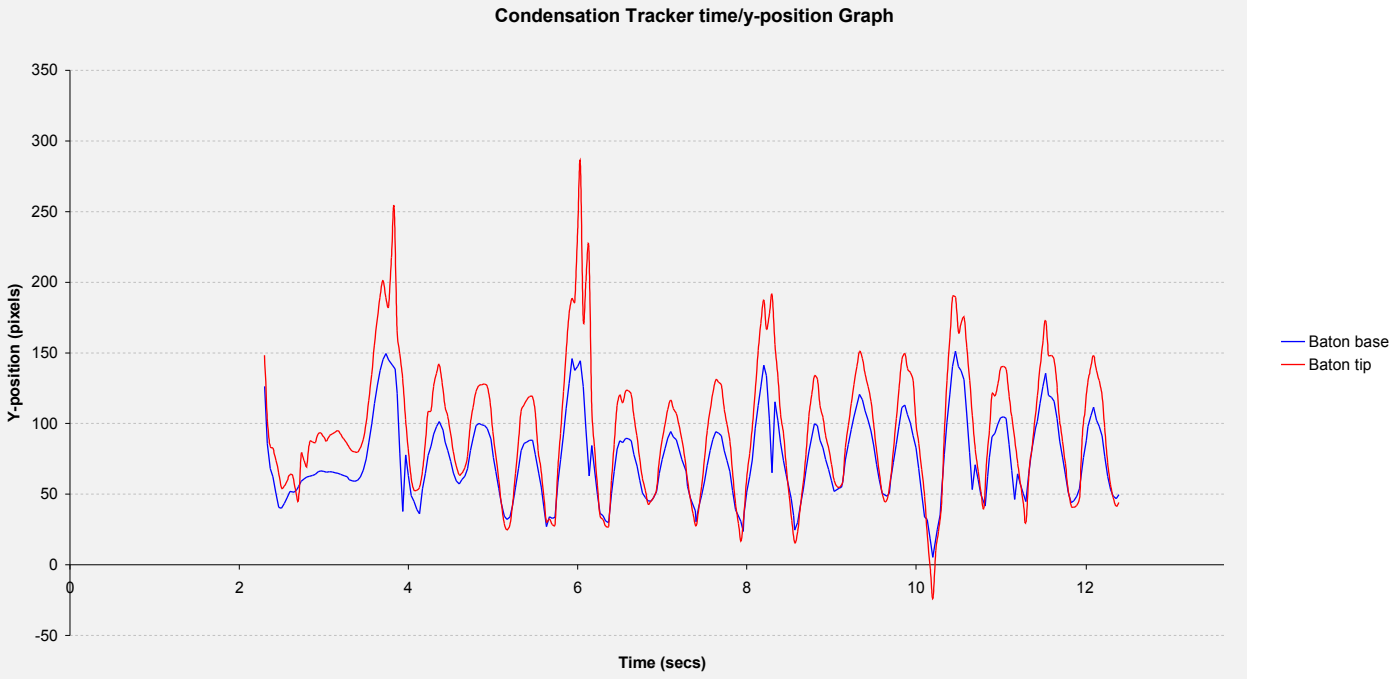
Fig. III.3-1 *– Graph showing the change in the baton's height with respect to time when beating 3 beats per bar.*



This would be fine if the tracker's output really did look like fig. III.3-1. All we'd need to do to detect the half beats in this case is to check when the vertical component of the bat-

on's velocity changes direction. However in reality the tracker's output tends to be very noisy, as fig. III.3-2 shows.

Fig. III.3-2 *– Graph showing an example of the condensation algorithm's output. The blue line at the bottom is the vertical trajectory of the baton's base, and the red line at the top is the vertical trajectory of its tip.*



The naïve method of detecting beats by looking for changes in the vertical direction of the baton would have produced lots of false positives in this case, due to the local peaks caused by noise. The data clearly needs to be filtered.

Our approach to filtering the data is based on the observation that even when noise produces false peaks in the vertical trajectories, the general shape of the true peaks is still preserved (i.e. the signal to noise ratio is high). This allows us to define properties that a true peak in the data at time t should satisfy that a false peak due to noise generally shouldn't.

We defined the following function:

$$Q_L(t) = \frac{1}{|\Phi_{L,t}|} \sum_{s \in \Phi_{L,t}} \left\{ y_{0,s} + y_{1,s} - (y_{0,t} + y_{1,t}) \right\} \qquad \text{(Eq. III.3.1.1)}$$

Where:

$y_{0,t}$, $y_{1,t}$ are the y-ordinates of the baton's tip and base at time t.

$\Phi_{L,t} \subseteq \{ s ;\ \ t < s < t + L \}, L \in \mathbb{N}, L > 0$

This function gives the average vertical displacement of the baton's tip and base between time t and time $t+L$ for some arbitrary L. $\Phi_{L,t}$ defines the times at which the sampled baton positions given by the tracker were non-NULL. By choosing a suitable

value for L, the polarity of this function gives us the direction of the baton's vertical motion whilst filtering out the false peaks that occur due to noise, and so we can use changes in its polarity as an indication of the beat times. Notice that for the special case of $L = 1$, detecting beats with this function is very similar to beat detection by the naïve method mentioned above, except that this function takes into account information from both the base and the tip of the baton. The negative point about this function however is that increasing L increases the delay between the time at which a beat occurs and the time at which it is detected. Setting L to 5 seems to work quite well at a frame rate of 30 frames/sec however. The delay is $^1/_6$ seconds in this case, which is acceptable.

III.3.2 Beat Prediction

A beat detector cannot determine when a half beat will occur until after it has occurred. By this point a musical response from the system would be too late. However by anticipating when the next beat will occur, the system can prepare its response in advance, allowing it to potentially respond in synchronisation with the conductor.

In order to predict when the next beat is going to occur, we need to model the temporal changes of the music. One simple way to do this is to make a similar assumption to that made for the motion model described in section III.2.2.1, i.e. we assume that the rate at which the half beat rate is changing stays approximately constant across consecutive half beats.

Define t_i as the time of the i^{th} half beat, and let t_{m-1} be the observed time, in seconds, of the last half beat. The half beat rate, b_{i-1} half beats/sec, of half beat i-1 can be approximated by $b_{i-1} \approx 1/(t_i - t_{i-1})$. Our assumption can be expressed in these terms as:

$$\frac{b_{m-1}-b_{m-2}}{t_{m-1}-t_{m-2}} \equiv (b_{m-1}-b_{m-2})b_{m-2} \approx \frac{b_{m-2}-b_{m-3}}{t_{m-2}-t_{m-3}} \equiv (b_{m-2}-b_{m-3})b_{m-3}$$

This implies that the duration d_{m-1} of half beat m-1 can be estimated as:

$$d_{m-1} \equiv \frac{1}{b_{m-1}} \approx \frac{b_{m-2}}{(b_{m-2}-b_{m-3})b_{m-3}+b_{m-2}^2} \qquad \text{(Eq. III.3.2.1)}$$

Using this, we can predict that half beat m will occur at time:

$$t_m = t_{m-1} + d_{m-1}$$

As with the motion model, this allows gradual changes of tempo to be predicted, but it is unable to predict sudden changes of tempo. Notice however that d_{m-1} is negative if:

$$(b_{m-2}-b_{m-3})b_{m-3}+b_{m-2}^2 < 0$$
$$\Rightarrow b_{m-3}b_{m-2} < b_{m-3}^2 - b_{m-2}^2 = (b_{m-3}+b_{m-2})(b_{m-3}-b_{m-2})$$

As $b_{m-3}b_{m-2}$ is positive, this implies that d_{m-1} may be negative (depending on the values of b_{m-3} and b_{m-2}) when $b_{m-3} > b_{m-2}$, i.e. when the tempo is decreasing.

An alternative method that doesn't give negative values for d_{m-1} when the tempo is decreasing is to assume that the rate of change of the duration of each half beat stays constant across consecutive half beats, i.e.:

$$d_{m-1} - d_{m-2} \approx d_{m-2} - d_{m-3}$$

$$\Rightarrow d_{m-1} \approx 2\,d_{m-2} - d_{m-3} \qquad \text{(Eq. III.3.2.2)}$$

This however has the opposite problem of producing negative values for d_{m-1} when the tempo is *increasing*. Our system allows for two alternative solutions to these problems. One is to just use one of the above equations, and assume a constant tempo across consecutive half beats whenever a negative value for d_{m-1} is predicted. The other is to use eq. III.3.2.1 when the tempo is increasing and eq. III.3.2.2 when the tempo is decreasing.

PART IV: EVALUATION

The following sections discuss tests we performed on our system together with the results we obtained. In all of our tests we used videos captured at 30 frames/sec with a Philips TouCam Pro II webcam at a resolution of 320x240. Unlike certain other cameras, this camera had the advantage of permitting us to disable its built-in automatic compensation for changes in lighting, meaning we didn't have to take changes in the camera's configuration into account when evaluating the performance of our system.

To make our tests reproducible, we performed our tests on video files captured from this device rather than performing the tests in real time directly from the camera. The disadvantage with this is that it didn't allow us to explore the effect of the noticeable delay between the time of the user's movement and the time at which the camera captured that movement. This could be investigated as part of a future project.

IV.1 Shape Recognition-Based Tracker Evaluation

As we didn't have any way of measuring the ground truth accurately, we were unable to give a quantitative evaluation of the accuracy of the shape recognition-based tracker. Instead we will make the following comments about it:

- By inspection, the tracker generally seems to be able to track the coordinates of the baton's end points to within approx. 5 pixels of their observed locations.
- The accuracy to which it tracks the baton is largely dependent on the following two factors:

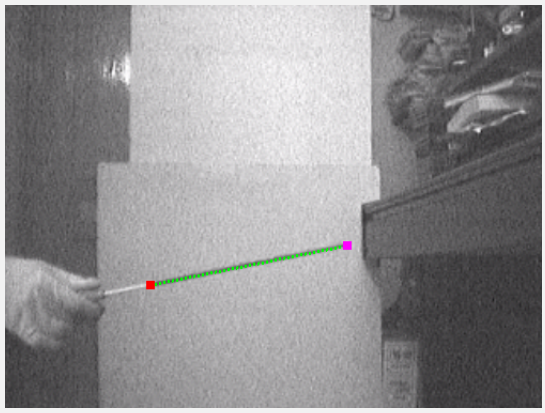
  1. **The quality of the estimated background image.** Increasing the number of frames over which the background image is accumulated tends to reduce the amount of noise that escapes the background subtraction process.

  2. **The contrast between the baton and the background.** Regardless of how much noise the background subtraction process filters out, the tracker can never track the baton accurately when the whole baton or part of the baton is indistinguishable from the background. In the former case the tracker will fail completely. In the latter case it may track the largest part of the baton that is visible.

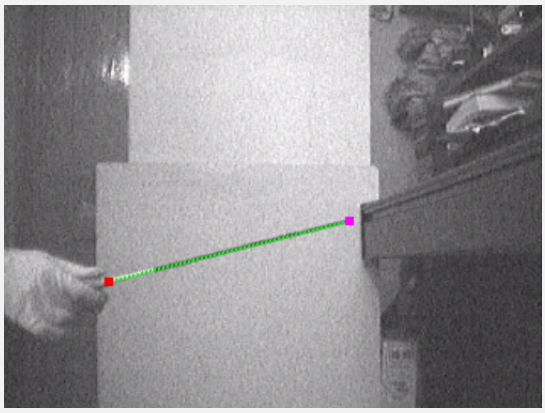
The contrast between the baton and the background generally decreases as the baton moves further away from the camera. This seems to be due to the fact that the baton has such a thin projection onto the camera that beyond a certain distance its projection onto the camera is "overwhelmed" by ambient light. The only way around this with the equipment we had was to keep the camera close to the user. This had the unfortunate effect of constraining the user's movement to the small area that is visible within the camera. A better solution may have been to film the user at a higher resolution with a camera that has a better image quality, however filming at too high a resolution would have had an adverse effect on the system's performance.

- The tracked points sometimes fluctuate due to changes in the illumination of the baton as it moves. We painted the top 25.5 cm of the baton black, leaving 7cm at the bottom white to improve the baton's contrast against the white background we used to test the system (as shown in fig. III.2-1). Despite this, the tracker is sometimes unable to distinguish between the white part of the baton and the black part, which causes its estimate of the position of the baton's base to jump, as shown in fig. IV.1-1. This problem can generally be solved for a single image by setting the segmentation threshold to a lower value, however it is difficult to find a value that works well across all frames.

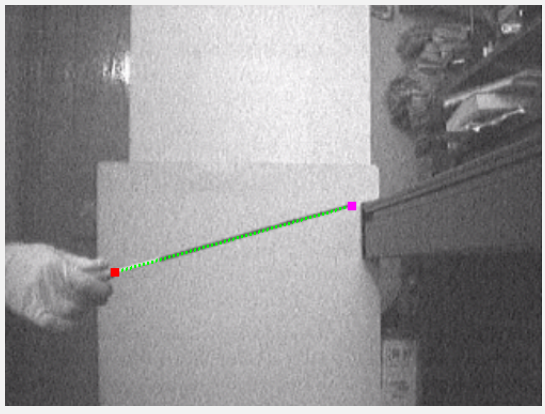
Fig. IV.1-1 *– A sequence of frames showing a fluctuation in the tracked position of the baton's base.*



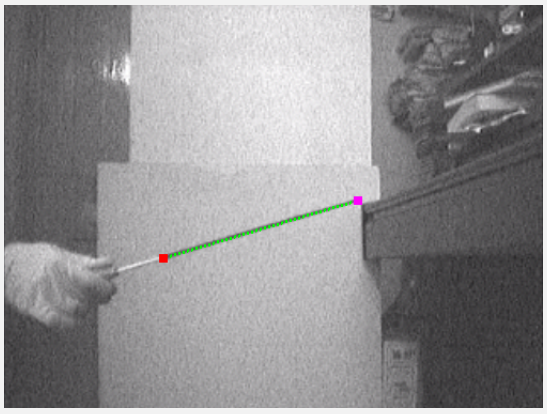
(Frame 194)



(Frame 195)



(Frame 196)



(Frame 197)

IV.2 Condensation Tracker Evaluation

We tested the C.A.-based (Condensation Algorithm) tracker's accuracy by comparing its performance on a particular video to a benchmark set by the S.R.-based (Shape Recognition) tracker for that video. Testing the accuracy of the tracker in this way is only useful when we know that the S.R.-based tracker's output is accurate to within a given error bound. As we had no way of measuring this error bound, we had to judge the S.R.-based tracker's accuracy by inspection. The following test is based on the video `Indeo_test01.avi`, which will be made available on our project web page in due course.

We ran the S.R.-based tracker on the test video with the following parameters:

Min. standard deviation of pixels about axis:	0.4
Max. standard deviation of pixels about axis:	1.8
Min. segment length:	25.0
Max. segment length:	150.0
No. frames for background accumulation:	50
Background subtraction threshold:	0.007
Segmentation threshold:	0.269

By inspection, the S.R.-based tracker seemed to track the y-ordinates of the baton's end points very accurately in the test video. This resulted in the trajectories shown in fig. IV.2-1. They are smooth for the most part, except for a few points around the 3 second mark and the 6.5 second mark. To fully-appreciate this however, the reader would need to observe the tracker for himself.

The S.R.-based tracker's tracked paths for the x-ordinates of the baton's end points were not as accurate however (see fig.). The trajectory of the base is particularly noisy due to the fluctuations we discussed in the previous section. In view of this discrepancy, we only compared the tracked y-ordinates of the C.A.-based tracker to those of the S.R.-based tracker.

We defined the following error function to determine how close the C.A.-based tracker's output baton state in any given frame was to that of the S.R.-based tracker in the same frame:

$$error(B_S, T_S, B_C, T_C) = \sqrt{\left\|B_S - B_C\right\|^2 + \left\|T_S - T_C\right\|^2}$$

Where:

B_S, T_S are the base and tip positions respectively tracked by the S.R.-based tracker.

B_C, T_C are the base and tip positions respectively tracked by the C.A.-based tracker.

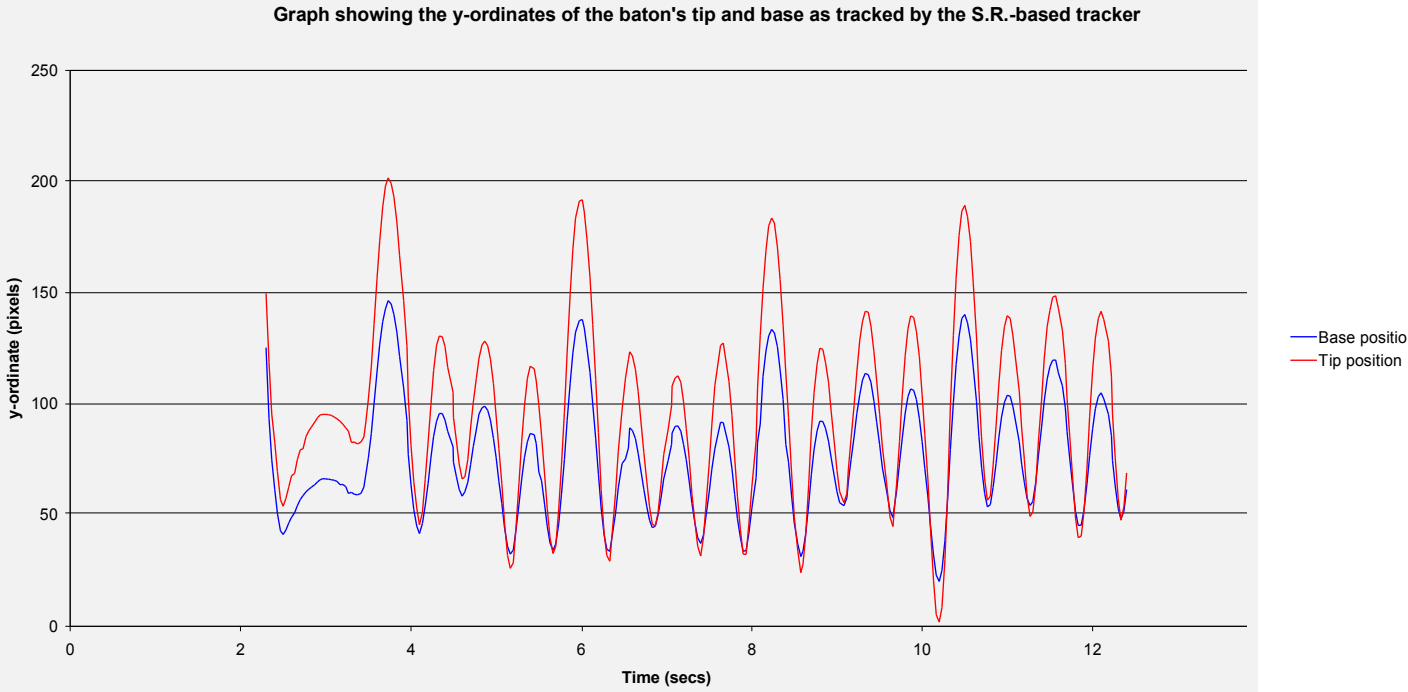
For our test we wanted to investigate the effect that changing the size of the sample set would have on the condensation algorithm's accuracy. We set the parameters of the S.R.-based tracker that it uses to calculate the importance function to the same values as those given above. We set the remaining parameters to the following values:

Edge detection threshold:	0.047
Hypothesis rejection threshold, R:	0.001
No. search lines on each side of the hypotheses:	10
Probability of using importance sampling when not reinitialising, I:	0.5
Search window size at base and tip:	10
σ_M***:***	10.0
Seed for the uniform distribution sampler used to draw samples from S_{t-1}***:***	1
Seed for the Gaussian distribution sampler used by the motion model to draw samples for the baton's base:	1304
Seed for the Gaussian distribution sampler used by the motion model to draw samples for the baton's tip:	65403

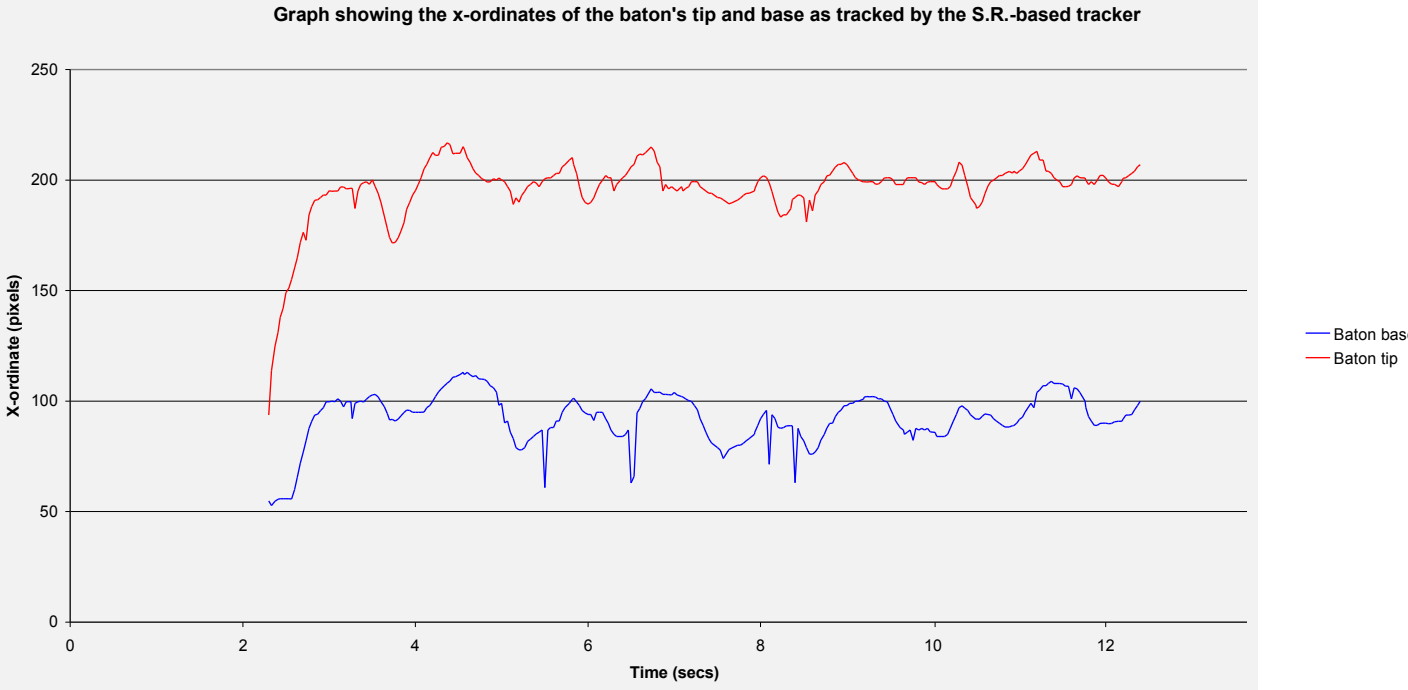
We performed the test by varying the sample set size from 5 samples to 500 samples, keeping the other parameters the same. For each sample set size, we tracked the baton over the 1202 frame test video and calculated the error in each frame. Fig. IV.2-2 summarises the average error for each sample set size. The full results will be made available on our project web page.

Our results show a very high average error of 22.28236 with a sample set of size 5, however it drops rapidly as we increase the sample size from 5 to 40. Beyond 40 samples, the average error seems to have converged to about 2.3. The significance of its convergence to this particular value is questionable, as we do not know the error bound of the S.R.-based tracker, so this limiting value may be influenced by the fact that our error function is biased towards the S.R.-based tracker.

Fig. IV.2-1 *– Graph (a) shows the y-ordinates of the tip and base of the baton as tracked by the S.R.-based tracker during the first 12.4 seconds of the test video. The red line at the top is the trajectory of the baton's tip, and the blue line at the bottom is the trajectory of its base. Graph (b) shows the x-ordinates from the same data set.*



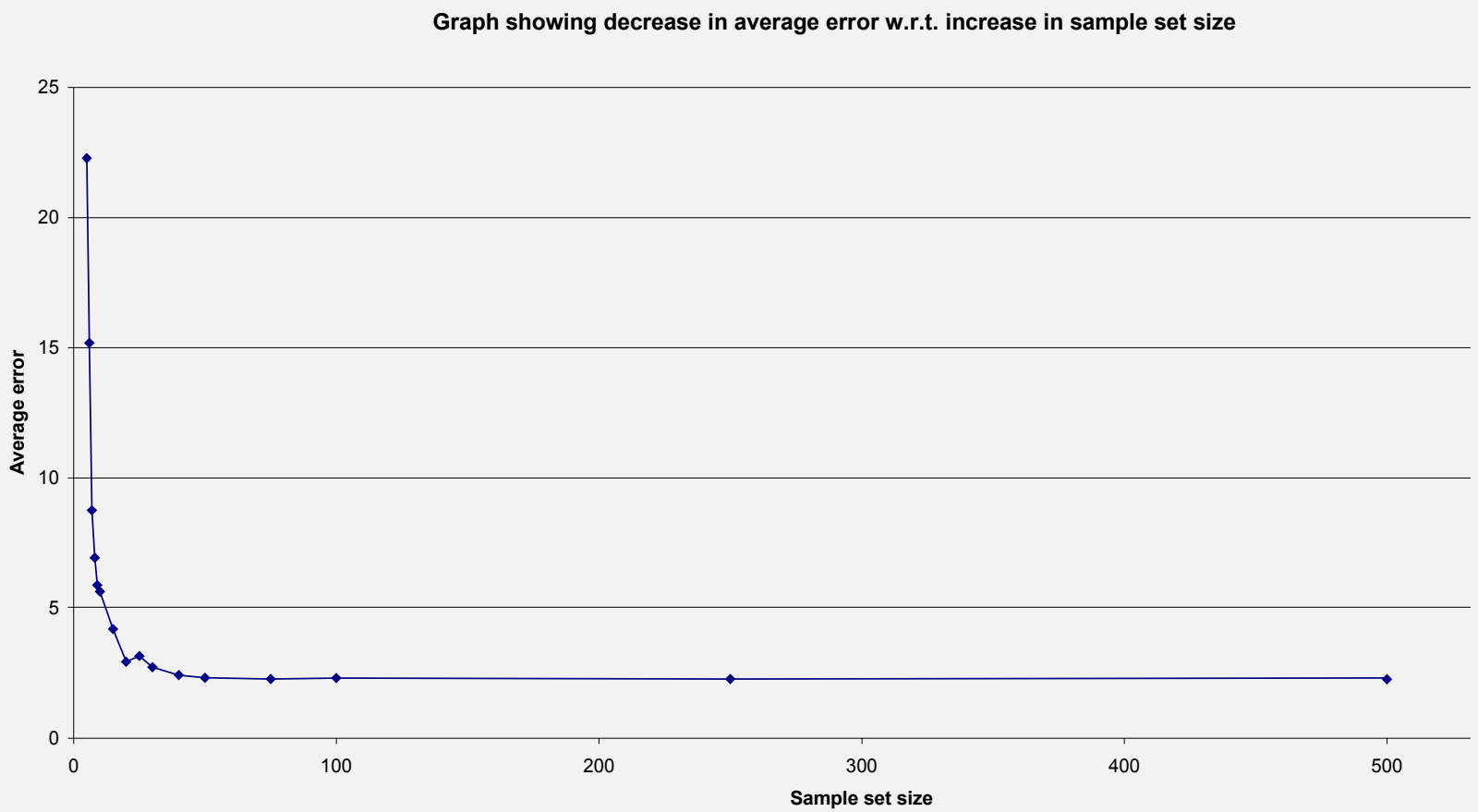
(a)



(b)

Fig. IV.2-2 *– A summary of the results from our test of the relationship between the condensation algorithm sample set size and the average tracker error.*

No. samples	**Avg. error**
5	22.28236
6	15.19094
7	8.75679
8	6.934342
9	5.872108
10	5.627815
15	4.189724
20	2.945111
25	3.162355
30	2.726361
40	2.42364
50	2.329063
75	2.272791
100	2.314119
250	2.276203
500	2.258517



Part V: Conclusion

In conclusion, we found the condensation algorithm to be effective in tracking the baton to an acceptable average degree of error. However from our tests we found that the algorithm tends to reinitialise the entire sample set in one go from the importance function very frequently. With a set of 5 samples, the algorithm would sometimes reinitialise all of them up to 8 times per second. With a larger set of 500 samples, the algorithm rarely seemed to run for much longer than 1 second before reinitialising all of the samples.

This is almost certainly due to the inadequacy of our motion model. As noted earlier, the motion model should have taken the covariance between the velocity variables into account. A good motion model should also take into account the expected time of the next beat, as this gives an indication of when the baton is going to change direction. The motion model could have used this information together with a prior model of the motion of the conductor's baton for different time signatures (as shown in fig. II.2-2) to predict the baton's motion more accurately.

So although our implementation of the condensation algorithm can give good results, its high dependence on reinitialising itself from the S.R.-based-tracker-derived importance function makes it highly sensitive to all of the weak points of the S.R.-based tracker discussed in section III.2.1.5. This makes our implementation of it unsuitable for use in any environment where the S.R.-based tracker's background subtraction phase is unable to remove persistently distracting background features that it would otherwise mistake for the baton. An example of such an environment would be one where the lighting is not static.

V.1 Future Work

There is a significant amount of further work that would need to be done in order for our system to be turned into a fully-fledged consumer product, much of which involves a number of interesting areas of further research. The following is a list of some of these extensions:

- First and foremost a more accurate motion model needs to be implemented so as to reduce the dependence of the C.A-based tracker on the S.R.-based tracker. Doing so should improve its ability to track the baton in a cluttered environment.
- The S.R.-based tracker's output could be improved, as mentioned before, by

joining collinear segments that are close to each other. This may allow the conductor to stand further away from the camera without degrading the accuracy of the tracker's output, allowing him to move more freely.

- The simple beat prediction methods we used are unable to cope with sudden changes of tempo. One way to model such changes would involve using a machine learning algorithm (such as the one suggested in [CR1]) to learn how the conductor tends to pull the tempo about by rehearsing with him. Such an algorithm would be able to automatically calculate the error in its predictions by comparing its predicted times to the times indicated by the beat detector. A suitable learning algorithm could then be applied to improve the system's predictive accuracy based on these errors.

- The quality of the MIDI sounds generated by the sound card wouldn't be very satisfying for a musician to work with. A much better approach to this important aesthetic aspect of the project is that described in [JBWSMM1], where real recordings of an orchestra playing are retimed using the Fast Fourier Transform. In that project the researchers pre-calculated a set of retimed audio tracks, and then played back the one that was closest to the true tempo of the conductor. An interesting area of research would be an investigation into whether or not this can be done in real time, so that the conductor's tempo can be matched more accurately.

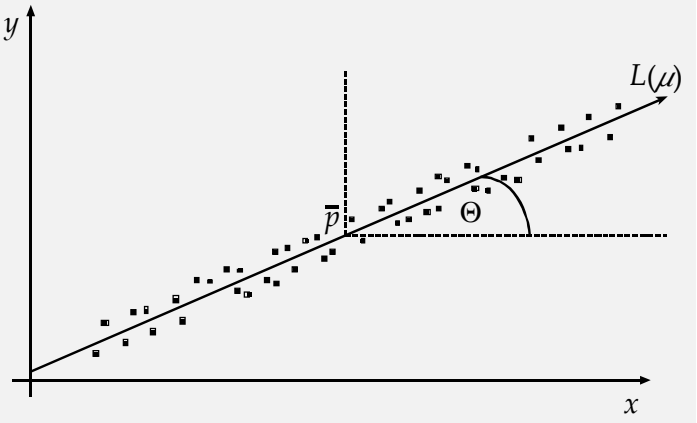
- The system should ideally be extended to interpret a wider range of performance directions that a conductor may give. This would involve implementing a more sophisticated gesture analysis component, and may require interpreting other aspects of the conductor's actions besides the motion of his baton.

- To make the system more useful as a practise tool, features could be added to the system that allow it to analyse the conductor's performance and give him feedback on the quality of his conducting, possibly suggesting areas that the conductor needs to work on.

APPENDIX A

Proof of Theorem II.3.2.2-1

***Fig. A-1**: Graph showing a 2D data set and its central axis, $L(\mu;\bar{p},\theta)=\bar{p}+\mu(\cos\theta,\sin\theta)$, as solved by polar co-ordinate linear regression.*



Given a set P of N 2D data points, the aim of polar linear regression is to find a line of the form $L(\mu;\bar{p},\theta)=\bar{p}+\mu(\cos\theta,\sin\theta)$ that minimises the sum of the squared perpendicular distances between itself and the data points.

Let $p_i=(p_{ix}, p_{iy})$ be the i^{th} data point.

The proof of theorem II.3.2.2-1 begins with the result from principal component analysis that the principal component of a data set passes through its mean. From this we can write down:

$$\bar{p} = (\bar{p}_x, \bar{p}_y) = \frac{1}{N}\left(\sum_i p_{ix}, \sum_i p_{iy}\right)$$

We will make use of the following trigonometric identities in the rest of the proof:

$$\cos^2\Theta+\sin^2\Theta \equiv 1 \qquad \text{(Eq. A-1)}$$

$$\cos 2\Theta \equiv \cos^2\Theta-\sin^2\Theta \qquad \text{(Eq. A-2)}$$

$$\sin 2\Theta \equiv 2\cos\Theta\sin\Theta \qquad \text{(Eq. A-3)}$$

$$\cos^2\Theta \equiv \frac{1+\cos 2\Theta}{2} \qquad \text{(Eq. A-4)}$$

Let:

$v_i = p_i - \bar{p}$ for all $0 \le i < N$.

$L = L(1) - \bar{p} = (\cos\theta, \sin\theta)$

By rearranging $v_i \cdot L$ for arbitrary i and then using Pythagoras' theorem and eq. A-1, the squared perpendicular distance from p_i to L, ϵ_i^2 is:

$$\epsilon_i^2 = v_i \cdot v_i - (v_i \cdot L)^2 = v_i \cdot v_i - (v_{ix}^2 \cos^2\theta + 2 v_{ix} v_{iy} \cos\theta \sin\theta + v_{iy}^2 \sin^2\theta) \quad \text{(Eq. A-5)}$$

And so we can define the sum of the squared perpendicular errors, $R(\theta)$ as:

$$R(\theta) = \sum_i \epsilon_i^2 \quad \text{(Eq. A-6)}$$

To find (cos θ, sin θ) that minimises $R(\theta)$, we equate the derivative to 0 and find the minimum:

$$\frac{d(R(\theta))}{d\theta} = -2 \sum_i \left[\sin\theta \cos\theta (v_{iy}^2 - v_{ix}^2) + v_{ix} v_{iy} (\cos^2\theta - \sin^2\theta) \right] = 0 \quad \textit{at turning points} \quad \text{(Eq. A-7)}$$

$$\Rightarrow (\cos^2\theta - \sin^2\theta) \sum_i v_{ix} v_{iy} + \sin\theta \cos\theta \sum_i (v_{iy}^2 - v_{ix}^2) = 0$$

and

$$\Rightarrow (\sin^2\theta - \cos^2\theta) \sum_i v_{ix} v_{iy} - \sin\theta \cos\theta \sum_i (v_{iy}^2 - v_{ix}^2) = 0$$

Hence if (cos θ, sin θ) = (cos ϕ, sin ϕ) is a solution, (cos θ, sin θ) = (-sin ϕ, cos ϕ) is also a solution.

Let:

$$A = \frac{1}{2} \sum_i (v_{iy}^2 - v_{ix}^2)$$

$$B = \sum_i (v_{ix} v_{iy})$$

Using eq.s A-2 and A-3 we can rewrite eq. A-7 as:

$$0 = A \sin(2\theta) + B \cos(2\theta) \quad \text{(Eq. A-8)}$$

Which implies that the vectors (B, A, 0) and (cos 2θ, sin 2θ, 0) are perpendicular. Hence their cross product is proportional to a unique vector given by:

$$(0, 0, A\cos 2\theta - B \sin 2\theta) = \left(0, 0, \pm \|(\cos 2\theta, \sin 2\theta, 0)\| \cdot \|(B, A, 0)\| \sin\frac{\pi}{2} \right)$$

$$\Rightarrow A\cos 2\theta - B\sin 2\theta = \pm\sqrt{B^2+A^2},\ \ by\ eq.\ A-1$$

$$\Rightarrow \frac{A\cos 2\theta \pm \sqrt{B^2+A^2}}{B} = \sin 2\theta$$

$$\Rightarrow B\cos 2\theta + A\left[\frac{A\cos 2\theta \pm \sqrt{B^2+A^2}}{B}\right] = 0,\ \ by\ eq.\ A-8$$

$$= (\cos 2\theta)\left(B+\frac{A^2}{B}\right) \pm \frac{A\sqrt{B^2+A^2}}{B}$$

$$\Rightarrow |\cos 2\theta| = \left|\frac{A}{\sqrt{B^2+A^2}}\right|$$

$$\Rightarrow |\cos\theta| = \left|\sqrt{\frac{|\cos 2\theta|+1}{2}}\right|,\ \ by\ eq.\ A-4$$

$$\Rightarrow |\sin\theta| = \left|\sqrt{1-|\cos\theta|^2}\right|,\ \ by\ eq.\ A-1$$

Now we simply need to determine the signs of $|\cos\theta|$ and $|\sin\theta|$ and check whether we have a maximum point or a minimum point. As the derivative has two solutions, we need to check $(\pm|\cos\theta|, |\sin\theta|)$ and $(\pm|\sin\theta|, |\cos\theta|)$.

Let (cos Θ, sin Θ) be the directional vector that minimises $R(\theta)$.

$$(\cos\Theta, \sin\Theta) = \arg\min_{\omega\in\Omega} R(\omega)$$

Where:

$$\Omega \subseteq \mathbb{R}\times\mathbb{R} = \{(c,s),(-c,s),(s,c),(-s,c)\}$$
$$c = |\cos\theta|$$
$$s = |\sin\theta|.$$

QED.

BIBLIOGRAPHY

[AKCDL1] A. Kapadia, C. Lu, "**Random Number Generators**". http://www.cs.dartmouth.edu/~akapadia/project2/project2.html.

[AMM1] A. M. McIvor. "**Background Subtraction Techniques**". In *Proc. of Image and Vision Computing*, Auckland, New Zealand, 2000.

[CR1] C. Raphael, "**A Probabilistic Expert System for Automatic Musical Accompaniment**". In *Jour. of Comp. and Graph. Stats. vol. 10 no. 3*, p. 487-512, 2001.

[DB1] D. Buchla, "**Lightning II MIDI controller**". http://www.buchla.com.

[DM1] D. Murphy. "**Tracking a Conductor's Baton**". In *Proc. 12th Danish Conference on Pattern Recognition and Image Analysis*, 2003.

[DMTAKJ1] D. Murphy, T. H. Andersen and K. Jensen. "**Conducting Audio Files via Computer Vision**". In *Springer vol. 2915*, Gesture Workshop 2003, p. 529-540, 2004.

[JBWSMM1] J. Borchers, W. Samminger and M. Mülhäuser. "**Engineering a realistic real-time conducting system for the audio/video rendering of a real orchestra**". In *IEEE MSE 2002 Fourth International Symposium on Multimedia Software Engineering*, December 2002.

[JG1] Jonathan Goodman. "**Lecture Notes on Monte Carlo Methods. Chap 2: Simple Sampling of Gaussians**". Courant Institute of Mathematical Sciences, NYU, http://www.math.nyu.edu/faculty/goodman/teaching/MonteCarlo2005/index.html, August 2005.

[JLNS1] J. Lumley and N. Springthorpe. "**The Art of Conducting**". Rhinegold Publishing Limited, 1989.

[KF1] K. Fishkin, A. Glassner (ed.). "**Filling a region in a frame buffer**". In *Graphics Gems I*, Academic Press, 1990.

[MI1] M. Isard. "**Visual Motion Analysis by Probabilistic Propagation of Conditional Density**". D.Phil. Thesis, Oxford University, 1998.

[MLGGDW1] M. Lee, G. Garnett, and D. Wessel. "**An Adaptive Conductor Follower**".

In *Proc. Int'l Computer Music Conf. '92*, Int'l Computer Music Assoc., San Francisco, p. 454-455, 1992.

[MMTN1] M. Matsumoto and T. Nishimura, "**Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator**". In *ACM Transactions on Modelling and Computer Simulation*, vol. 8, issue 1, p. 3-30, 1998.

[MVM1] M. V. Mathews, "**The Radio Baton and Conductor Program, or: Pitch, the Most Important and Least Expressive Part of Music**". In *Computer Music Journal*, xv/4, p. 37-46, 1991.

[MWWR1] Mathworld, "**Convolution**". http://mathworld.wolfram.com/Convolution.html.

[MWWR2] Mathworld, "**Least Squares Fitting**". http://mathworld.wolfram.com/LeastSquaresFitting.html.

[NRC1] W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling. "**Numerical Recipes in C: The Art of Scientific Computing**". Cambridge University Press, October 1992.

[TITT1] T. Ilmonen and T. Takala, "**Conductor Following With Artificial Neural Networks**". Int'l Computer Music Conf. '99, Beijing, 22-28 Oct 1999.

[TMN1] T. M. Nakra. "**Inside the Conductor's Jacket: Analysis, Interpretation and Musical Synthesis of Expressive Gesture**". Ph.D thesis, MIT Media Lab, February 2000.

[WP1] Wikipedia, "**Importance sampling**". http://en.wikipedia.org/wiki/Importance_sampling.

[WP2] Wikipedia, "**Inverse transform sampling method**". http://en.wikipedia.org/wiki/Inverse_transform_sampling_method.

[WP3] Wikipedia, "**Cholesky decomposition**". http://en.wikipedia.org/wiki/Cholesky_decomposition.

[WP4] Wikipedia, "**Gestalt psychology**". http://en.wikipedia.org/wiki/Gestalt_psychology.